Libki Manual

Table of Contents

License

1. Introduction

1.1. What is Libki?

Libki is a Free Open Source cross-platform pc reservation booking and time management system designed to allow time limited access to computers on a network. Libki is ideally suited for use in locations where a controlled computing environment is paramount such as public access systems, libraries, school computer laboratories and more! It consists of two parts, the Libki server, and the Libki client.

Learn more about Libki by visiting the official Libki website, libki.org.

1.1.1. The Libki server

The Libki server utilizes a web-based administration system that can be accessed from any networked pc via a web browser. From the web administration one can create and delete users, change the amount of time left on an account, send or leave a message for a user, log out and/or ban users, and see which client computers are available and which are currently being used. In addition, the system has a public web interface for viewing which kiosks are available, which are unavailable, and to allow the placing of reservations.

1.1.2. The Libki client

The Libki client is cross-platform and runs on many operating systems including Microsoft Windows and Linux. The Libki client features a counter to let the user know how many minutes are left, and the ability to log off early. Any unused minutes can be used at any other computer running the Libki client. The Libki client can be set to log off from the operating system, or even reboot the computer when a user logs off from the Libki client.

1.2. Libki server recommendations

At the time of this writing, it is recommended to run Libki Server on Debian Linux, Ubuntu Server or via Docker.

Guides for deployment are available in the Installation section.

Libki Server is meant to work on all web browsers, but is most tested on Chrome and Firefox.

1.3. Making contributions

This manual is an ever-changing document and edits to the manual are welcome at any time. The canonical source for the Libki manual is https://github.com/Libki/libki-manual

If you have a suggestions or contribution for the manual, please follow the guidelines for contributing to the Manual.

2. Installation

2.1. One-step automatic install

Do you want to setup Libki as fast as possible and have a dedicated server? Then this is the choice for you. Just make sure your server is running Ubuntu 18.04, 20.04 or Debian 10 (Buster) and that you don't have a user named 'libki'.

```
wget -O- install.libki.org | bash
```

(By the way, that's the letter O, not the number 0.)

2.2. Alternative method

Not everyone wants to pipe a script directly from the web to bash. It prevents you from reading the code you're about to run on your system. If you want to read it through first, this alternative should be for you:

```
git clone https://github.com/libki/libki-server.git
cd libki-server
bash install.sh
```

2.3. Docker

The official Libki Server Docker image needs to be connected to a database server, it does not contain a MySQL instance within it. Using a MySQL or a MariaDB container on the same Docker network is considered good practice.

The Libki Server image is available on Docker Hub.

This container requires some environment variables to be set in order to function correctly, the following is a mocked example of the environment variables needed:

```
LIBKI_INSTANCE=<INSTANCE> ①
LIBKI_DB_DSN=dbi:mysql:<DB_NAME>;host=<DB_HOSTNAME>;port=3306
LIBKI_DB_USER=<DB_USERNAME>
LIBKI_DB_PASSWORD=<DB_PASSWORD>
LIBKI_DB_HOST=<DB_HOSTNAME>
LIBKI_DB_PORT=3306
LIBKI_DB_DATABASE=<DB_NAME>
LIBKI_TZ=America/Denver ②
```

① Instance name, may be left empty for single-instance servers

② Find your timezome in the documentation here.

2.3.1. Cronjobs

The Libki cronjobs are not set to run inside the Docker container. Instead, it the cronjobs should be set up on the host machine.

Add the following to the root user's crontab or to a file in `/etc/cron.d/`.

```
* * * * * root docker exec <container-name> /app/script/cronjobs/libki.pl
0 0 * * * root docker exec <container-name> /app/script/cronjobs/libki_nightly.pl
```

3. Configuration

3.1. Settings

3.1.1. Time settings

Default time allowance per day

This setting determines the total number of minutes a user will have for the day by default.

The value of **Default time allowance per day** should be a whole number.

Default time allowance per session

This setting determines the total number of minutes a user will have for each *session* per day. So, if a system is configured with 120 minutes time allowance per day, and 30 minutes time allowance per sessions, a user will be able to log in 4 times per day for 30 minutes at a time.

The value of **Default time allowance per session** should be a whole number.

Default guest time allowance per day

This is the same as **Default time allowance per day**, except for guest accounts instead of regular users.

The value of **Default time allowance per day** should be a whole number.

Default guest time allowance per session

This is the same as **Default time allowance per session**, except for guest accounts instead of regular users.

The value of **Default guest time allowance per day** should be a whole number.

3.1.2. Automatic time extension

Libki has an automatic time extension system that allows users to remain logged into a client computer beyond the pre-determined number of minutes per session a user is allotted.

Extension length

The amount of time to automatically increase a user's session time by \(in minutes \).

The value of **Extension length** should be a whole number.

Extend time at

This setting controls at what point in time an extension length is triggered. A session will be extended when the user's session time drops below this number, in minutes.

The value of **Extend time at** should be a whole number.

Extend time unless

This setting determines if a user is eligible for an automatic time extension. It has two options:

- User's client is reserved
  - This choice prevents a time extension in the case that the user's client is reserved. Reservations for other client's are not taken into account.
- Any client is reserved
  - This choice prevents a time extension in the case that **any** client is currently reserved. If any client is reserved a time extension will not take place, even if the users's client is not currently reserved,

When extending time

This setting determines how minutes are added to a patron's account when an automatic time extension occurs. It has two options:

- Take minutes from daily allotment
  - This options moves minutes from the user's daily allotment of minutes to the user's session minutes. That means the user can continue using the client computer, but only up to his or her daily allotment of time.
- Don't take minutes from daily allotment
  - This option adds minutes to a users session "out of thin air". As such, it does not effect how many sessions a user will have per day.

3.1.3. Client behavior

Client reservations

This setting controls how clients may be used by users. There are three possibilities.

- First come, first served.
  - Allows a patron to log in to any client not currently being used.
- Reservation only.
  - Requires a patron to place a reservation for a client before logging in to it.
- First come first served, with option to place reservation.
  - A hybrid approach that allows patrons to log in to any client that is not currently in use or reserved.

The setting Display usernames controls if the user's username is displayed on the reserved client computer.

Client inactivity

These settings control the default automatic logout due to user inactivity.

These server settings can be overridden on a per-client basis. See the Client configuration for more details.

Client inactivity warning

Warn user that they will be logged out automatically after this many minutes of inactivity.

Client inactivity logout

Log user out automatically after this many minutes of inactivity.

Client registration update delay limit

When a Libki client launches, it periodically connects to the Libki server to register itself. If a client does not re-register itself within this number of minutes, the Libki server will remove it from the list of active clients.

The value of **Client registration update delay limit** should be a whole number.

Client login banners

The client banners are optional areas on the top and bottom of the Libki client login screen. They are functionally like to web browsers. As such, anything that is viewable in a web browser is viewable in the banner areas \(size permitting \).

Source URL

The URL for the image or html that you wish to display in the banner section.

Width

If the **Source URL** is an image, it can be forced to a specific width instead of using the image's actual width. Leave empty to use the image's actual width.

Height

This is the same as **Width** for the **Source URL** but for height.

3.1.4. Custom Javascript

Administration interface JavaScript

Add custom JavaScript for the admin interface in this text box.

Public interface JavaScript

Add custom JavaScript for the public interface in this text box.

3.1.5. Data retention

History anonymization

This setting controls how long user and client data is retained in an un-anonymized format. If this setting is left empty, the data will be kept indefinitely.

When anonymized, a particular user will be given a new random id on a per-day basis. In this way, statistics will still be able to track usage on a per-user basis per day, but will be unable to track per-user usage over any longer time period.

The value of **History anonymization** should a whole number.

History retention

This setting controls how long user and client data is retained for the purpose of generating statistics and checking usage history, in days. If this setting is left empty, the data will be kept indefinitely.

The value of **History retention** should a whole number.

Inactive user retention

This setting controls how long a user can be inactive before being automatically deleted, in days. If this setting is left empty, all users will be kept indefinitely.

The value of **Inactive user retention** should be a whole number.

This setting should be used to conform to GDPR if necessary.

3.1.6. Guest passes

Prefix for guest passes

The phrase that each guest pass username should start with. If left empty, the phrase "guest" will be used (e.g. guest1, guest2, guest3, etc).

This setting can be a word or short phrase, but should contain only letters and numbers. Avoid using spaces or special characters.

Passes to create per batch

If the *Multiple guests* button is used, this setting will control how many guest accounts are generated with each clock.

The value of **Passes to create per batch** should be a whole number.

Username label

The text in this field will be prepended to the guest username, (e.g. *"Username:"*).

Password label

This setting works the same as **Username label** but for the generated password instead of the username.

Guest pass CSS

This setting allows the batch guest passes to be styled with CSS. The default value for this setting is:

```
body { /* default body style emulates a pre tag */
    font-family: monospace;
    white-space: pre;
    display: block;
    unicode-bidi: embed;
}
.guest-pass { /* each username and password is in a guest-pass span */
    /* page-break-before: always; */ /* This will cause each pass to have a page
break, good for use with receipt printers */
}
.guest-pass-username {} /* span containing the username label and the username itself
*/
.guest-pass-username-label {} /* span containing the username label */
.guest-pass-username-content {} /* span containing the username itself */
.guest-pass-password {} /* span containing the password label and the password itself
*/
.guest-pass-password-label {} /* span containing the password label */
.guest-pass-password-content {} /* span containing the password itself */
```

3.1.7. ILS integration

Patron hyperlinks

Entering a url here will cause the username in the user's table of the web administration to become a hyperlink with the user's username at the end. For example, *http://catalog.koha.library/cgi-bin/koha/members/member.pl?quicksearch=1&searchmember=* will link to the Koha ILS's search function for the given username.

SIP configuration

Single Sign-on can with an ILS can be achieved via SIP2. Settings for the ILS SIP2 server can be stored in the *libki_local.conf* file or the **SIP configuration** setting.

To enable SIP authentication, you will need to edit your libki_local.conf and add a section like this:

```
<SIP>
    enable 1
    host ils.mylibrary.org
    port 6001
    location LIB
    username libki_sipuser
    password PassW0rd
    terminator CR
    require_sip_auth 0
    enable_split_messages 0
    no_password_check 0 ①
    fee_limit 5.00 ②
    deny_on charge_privileges_denied ③
    deny_on recall_privileges_denied ④
    deny_on excessive_outstanding_fines ⑤
    deny_on_field AB:This is the reason we are denying you ⑥
    category_field PC ⑦
    pattern_personal_name , ⑧
</SIP>
```

① If enabled, Libki won't validate the password given against the SIP server, any password will work

② Can be either a fee amount, or a SIP2 field that defines the fee limit (e.g. CC), delete for no fee limit

③ You can set SIP2 patron status flags which will deny patrons the ability to log in

④ You can set as many or as few as you want. Delete these if you don't want to deny patrons.

⑤ The full listing is defined in the SIP2 protocol specification

⑥ You can require arbitrary SIP fields to have a value of Y for patrons to be allowed to log in. The format of the setting is *<Field>:<Message>*.

⑦ Specify a SIP tag here that contains the user category in the SIP user lookup response

⑧ Specify a pattern for splitting lastname and firstname in personal name field(AE) in SIP response

An equivilent configuration set via YAML in the system settings would look like this:

```
enable: 1
host: ils.mylibrary.org
port: 6001
location: LIB
username: libki_sipuser
password: PassW0rd
terminator: CR
require_sip_auth: 0
enable_split_messages: 0
fee_limit: 5.00
deny_on:
  - charge_privileges_denied
  - recall_privileges_denied
  - excessive_outstanding_fines
deny_on_field: "AB:This is the reason we are denying you"
category_field: PC
pattern_personal_name: ,
```

The SIP section requires the following parameters:

- enable: Set to 1 to enable SIP auth, 0 to disable it.
- host: The SIP server's IP or FQDN.
- port: The port that SIP server listens on.
- location: The SIP location code that matches the sip login.
- username: The username for the SIP account to use for transactions.
- password: The password for the SIP accouant to use for transactions.
- terminator: This is either CR or CRLF depending on the SIP server. Default is CR
- require_sip_auth: Does this SIP server require a message 93 login before it can be used? If so this should be set to 1 and the username/password fields should be populated. This should be set to 1 for Koha.
- enable_split_message: IF thie server supports split messages you can enable this. This should be set to 0 for Koha.
- fee_limit: As notated, this can be a set number or a SIP field to check. If the fee limit is exceeded, the user login will be denied.
- deny_on: This can be repeated to deny logins based on the patron information flags detailed in the SIP2 protocol specification.
- deny_on_field: This can be repeated to deny logins if the Specified field does not have a value of "Y". ==== LDAP

Single Sign-on with other systems can be achieved via LDAP. Settings for LDAP server are currently stored in the *libki_local.conf* only, though setting support is expected soon.

Make sure the URL beings with http:// or https:// as necessary.

3.1.8. Print management

Printer configuration

Refer to the main Print Management section for details.

3.1.9. Reservations

Reservation timeout

The amount of time \(in minutes \) that a user has to log into his or her reserved computer. If the user does not log in within the specified time limit, the reserved client will become available again.

The value of **Reservation timeout** should be a whole number.

Display usernames

This setting determines if a Libki client that is reserved and waiting will display the username of the person it is waiting for.

3.1.10. Terms of service

This setting allows a library to add *terms of service* for use of computers running the Libki client. Simply adding text of your terms of service in the textbox will cause the terms to be displayed to any person logging into a Libki client. If the person chooses *yes* the login will proceed as usual. If the person chooses *no* the login screen will be reset.

3.1.11. User settings

User categories

Add your list of user categories here as a list in YAML syntax.

Example:

```
---
- 'Category 1'
- 'Category 2'
- 'Category 3'
```

If Single Sign-on via SIP integration has been set up, and the SIP configuration for `category_field` has been set up, values from the SIP field will be automatically added to this list as the appear for the first time via SIP.

3.1.12. Advanced settings

Advanced rules

In this section, advanced rules can be set up to deterimine a number of behaviors based on various criteria.

For example:

```
-
  criteria:
    user_category: MyCategory1
    client_location: MyLocation1
  rules:
    session: 25
    guest_session: 20
-
  criteria:
    client_location: MyLocation1
  rules:
    session: 20
    guest_session: 20
-
  criteria:
    user_category: MyCategory1
  rules:
    session: 15
    guest_session: 20
-
  rules:
    session: 10
    guest_session: 20
```

Rules should be list from most specific to least specific. If a criteria is missing in a rule, then it is not used as part if the matching process.

For example, let's say we are attempting to log in as a patron with the category `MyCategory1`, and a client with the location `MyLocation2`.

- The first rule matches category, but not location, so it is skipped.
- The second rule only checks location, which is not a match, so it is skipped.
- The third rule only checks category, which matches, so it is used!
- The last rule has no critea, so it matches on **everything**

If there is no catch-all rule like this, Libki will fall back to the default time allowance settings from the `Time settings` section.

Even if a rule should match, it will be skipped if it doesn't contain the 'subrule' we are looking for

For example, take the following:

```
-
  criteria:
    client_location: MyLocation1
  rules:
    session: 15
-
  rules:
    session: 10
    guest_session: 20
```

Assume a guest is logging in to a `MyLocation1` client.

- The first rule will be skipped because guest_session is not defined in this rule.
- The second rule will be used because it matches all and **does** have guest_session defined.

Rules can specify multiple values for a given criteria

For example:

```
-
  criteria:
    user_category:
      - MyCategory1
      - MyCategory2
    client_location: MyLocationA
  rules:
    session: 25
    daily: 50
    guest_session: 20
    guest_daily: 45
```

The above example would apply to any user of category `MyCategory1` **or** `MyCategory2` logging into a client at location `MyLocationA`.

That is, this rule will apply to all the values in the list for that criteria.

Another way to think of it is that all criteria are AND'ed, and all the values for a criteria are OR'ed.

```
-
  criteria:
    user_category:
      - Cat1
      - Cat2
    client_location:
      - LocA
      - LocB
  rules:
    session: 25
    daily: 50
    guest_session: 20
    guest_daily: 45
```

Would be expressed in SQL like: `SELECT rule WHERE ( user_category = Cat1 OR user_category = Cat2 ) AND ( client_location = LocA OR client_location = LocB )`

Rule options

The list of all possible criteria include:

- `client_location`
- `client_name`
- `user_category`

The list of all possible rules include:

- `session`: The number of minutes per session a registered user is given
- `guest_session`: The number of minutes per session a guest user is given
- `daily`: The daily allotment of minutes to be given to a registered user for the day
- `guest_daily`: The daily allotment of minutes to be given to a guest user for the day
- `no_reservation_required`: Even if "Client reservations" is set to "Reservation only", allows a login without a reservation. The value should be '1' to enable.

3.2. Closing hours

Closing hours are a way to prevent users from starting a session that will be cut short by the closing of the location he or she is at. Closing hours can be set on a site-wide basis, or on a per-location basis. If a given location has no closing hours set, that location will use the *All locations* closing hours.

3.3. Print management

Print management in Libki can be done using two different backends: Google Cloud Print and CUPS.

3.3.1. Configuring your server with Google Cloud Print

To set up print management, first set up your printers in Google Cloud Print. Next, generate a client id and secret. Finally, enter your configuration in the **Printer configuration** setting as YAML. The code block below is an example configuration with two printer profiles for a single printer (one color, one monochrome).

Finding your cloud printer id:

1. Set up the printer for Google Cloud Print
2. Browse to https://www.google.com/cloudprint/simulate.html
3. Use the Printer Search API to search for you printers, or browse to https://www.google.com/cloudprint/search directly
4. Locate your printer in the results, find the "id" field, it should look something like `id: "ed4ddb78-dc03-8574-8687-be3995df8cd4"`

Getting your client id and client secret

1. Get oAuth2 Credentials
2. Browse to https://console.developers.google.com/
3. Enable Cloud API
4. Create credentials for OAuth 2 type="Other"
5. Save the ID and secret for use in the Libki print configuration

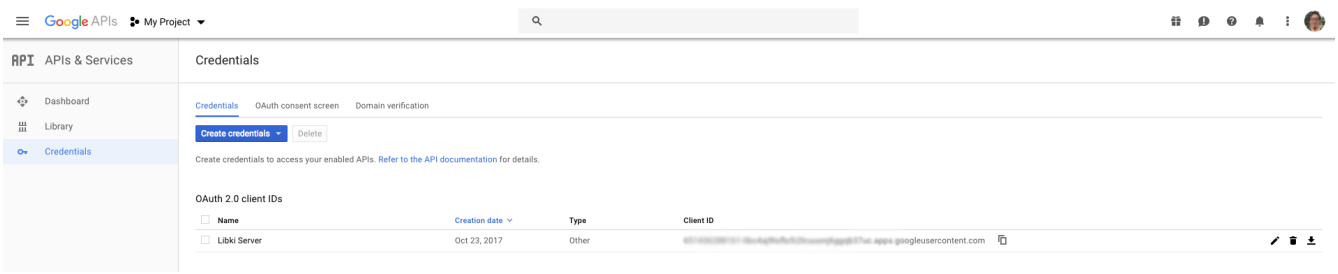

Figure 1. Google Developer Console

Setting up your print management configuration

You can use the configuration below as a template, simply replace the `client_id`, `client_secret`, and `google_cloud_id` field values with your own.

Note the example has two instances of the same printer installed, one for printing in color, and one for monochrome.

```
google_cloud_print:
  client_id: 893746288161-libc4aj9loitf5i2lcuuonj6ggqb37uc.apps.googleusercontent.com
  client_secret: dEjNmggj-PS9_LnvP92jIYu3

printers:
  color:
    type: google_cloud_print
    google_cloud_id: d4355eb9-5b5b-3982-1492-9a1245298409
    name: color
    ticket:
      color:
        type: STANDARD_COLOR

  monochrome:
    type: google_cloud_print
    google_cloud_id: d4355eb9-5b5b-3982-1492-9a1245298409
    name: monochrome
    ticket:
      color:
        type: STANDARD_MONOCHROME
```

Authorizing your server to use the Cloud Print API

1. Run `script/administration/enable_google_cloud_print.pl`
2. Open the URL given in a web browser
3. Authorize your account
4. Copy and paste the code you are given back into the script
5. Hit enter
6. You should recieve the message `Session stored.` if everything was successful.

3.3.2. Configuring your server with CUPS

To set up print management using CUPS, you need to know the name of the CUPS server (or its IP address), the username used to conect to that server and the name of the printer that will be set up.

Setting up your print management configuration

You can use the configuration below as a template, simply replace the `server` and `username` field values with your own.

```
cups:
      server: localhost
      username: print

printers:
  PDF:
    type: cups
    name: PDF
```

3.3.3. Configuring your clients

After you've set up your server for print management, you will then need to configure your clients as well. To accomplish this, you must edit the `Libki Kiosk Management System.ini` file. On Windows operating systems, this file is most often located in `C:\ProgramData\Libki` but may be located elsewhere depending on your specific OS configuration. You will most likely need to edit this file as an Administrator.

Once you have located the file, you will need to add a new configuration block to the bottom.

```
[printers]
color="C:\\printers\\color"
monochrome="C:\\printers\\monochrome"
```

As you can see, the printers match the `name:` fields defined print management configuration for the server. You will also need to ensure those directories exist on the client computer. Once the Libki client has been started, it will watch those directories for PDF files. When the client sees a file, it will uploaded it to the server with the matching printer name.

You can use any PDF print driver to print PDF file to these directories. A custom PDF print driver for Libki is in development, but not yet available.

4. Administration

4.1. Statistics & history

5. Desktop client

5.1. Windows

5.1.1. Download

The Windows wizard installer for the Libki client can be downloaded from https://github.com/Libki/libki-client/releases

Make sure to download the client version that matches your Libki server version.

5.1.2. Installer wizard

After downloading the client, run it and you will be presented with the installer wizard. Here you will be asked a series of questions about how you would like your client to be configured.

Scheme

Set this field to `https` if you have set up your Libki server behind an SSL-enabled proxy, set it to `http` otherwise.

Host

Set this to the ip address or FQDN for your Libki server.

Port

Set this field to the external port your Libki server is listening on.

Location

Set this field to the location you wish to label this client as being at. This is an arbitrary text field. If you are setting multiple clients to have the same location, make sure that you use the exact same spelling for all the clients.

For example, you may set the location to `Main Floor` or `Children's Room`.

Run only for this user

If you enter a Windows account username in this field, the Libki client will only launch when that Windows user is logged in to.

Run for all users but this one

This is the inverse of `Run only for this user`. If you set a Windows account username in this field, the client will not run for that user, but will for all other users.

Client name

Here you can name this client node, each client should have a unique username otherwise you will get unexpected behavior.

If you leave this field blank, Libki will user the client computers hostname as the Libki client's name.

If you set the value of this field to `OS_USERNAME`, the logged in Windows account username will be used as the client's name.

Logout action

This section will let you choose what happens when the Libki client logs out, either because the patron's time has run out, or the patron chooses to log out early.

Reboot

This option will trigger a full reboot of the client PC. This is useful when combined with the software Deep Freeze or Reboot Restore Rx which resets the state of a computer each time it is rebooted.

Log out of operating system

This option will trigger a logout of the Windows account that the client is currently logged in as. This is useful when combined with the software Clean Slate, which resets the state of a computer each time that the Windows account is logged out.

Nothing

This option leaves the host OS in the state it is currently in, and merely redisplays the login screen. This option is not recommended for production use. Any applications the user had open at the time the client was logged off of will still be open. This option is useful for testing and debugging.

Client disable password

Here you may enter a 'back door' password. At any time, if this password is entered into the Libki login screen with an empty username, it will cause the Libki client to close. This is useful in the event that a client loses connectivity with the server and is unable to verify logins.

Installation location

This is the path to which the Libki client will be installed. It is recommended to use the default installation path unless you have a specific reason to do otherwise.

5.2. Linux

In order to use the libki client on Linux, you must compile it from source. This guide is written with Debian based distros in mind (like Debian, Ubuntu, Linux Mint or Elementary OS).

5.2.1. Compilation

Install required packages:

```
sudo apt-get install git build-essential qtcreator qt5-default qttools5-dev-tools
libqt5webkit5-dev libqt5script5 qtscript5-dev
```

Download a copy of the libki client source.

```
git clone https://github.com/libki/libki-client.git
```

- Open Qt Creator, then select **Open File or Project** from the file menu.
- Select the **Libki.pro** file in the source folder you just downloaded.
- Click **Configure project** and then select **Build Project Libki** from the build menu.

5.2.2. Installation

Copy the compiled client from the debug folder that was created. It's located in the same directory as the libki-client source folder. FIXME Lookup proper folder name!

```
sudo cp libkiclient /usr/local/bin
```

Make it executable in case it for some reason isn't.

```
sudo chmod +x /usr/local/bin/libkiclient
```

Copy the configuration file and edit it to your liking. See Configuration options below for additional information.

```
nano ~/.config/Libki.ini
```

If the client is installed on another machine than the one compiling it, just some of the dependencies are needed.

```
sudo apt-get install libqt5webkit5 libqt5script5
```

5.2.3. Configuration

Linux Mint with Cinnamon

Contributed by Loidor

Bypass prevention

I'm running libkiclient through startup applications without a delay, and that launches it fast enough. Then I'm running a script I call demapper with a 2 second delay, because it isn't reliable with a shorter delay. That disables Alt and Super, so it's impossible to switch workspace, launch the start menu and opening the run console.

Code for demapper:

```
#!/bin/bash

xmodmap -e 'keycode 133='
xmodmap -e 'keycode 64='
```

Other bypass preventions:

- Disable the power button
- Disabling remote media popup (Nemo→Preferences→Behaviour)
- Replace the user shell with rbash chsh -s /bin/rbash USERNAME

I haven't encountered anyone terminal-savvy enough yet, but once I do I'll look into disabling kill, killall and pkill.

Autologin, backup, restore

I've disabled the screensaver, since I don't want the client to become locked, and have an autologin set in /etc/lightdm/lightdm.conf.

Together with this, I've got a backup/restore solution where backup copies /home/USERNAME to /opt/USERNAME. restore deletes /home/USERNAME and replaces it with /opt/USERNAME. Only root can run backup when needed (after changes), and restore is run on every logout through lightdm.conf.

backup code:

```
#!/bin/bash

if [[ $EUID -ne 0 ]]; then
   echo "This script must be run as root"
   exit 1
fi

rm -rf /opt/public
cp -a /home/public /opt/public
echo "Backup klar."
```

restore code:

```
#!/bin/bash

rsync -qrpog --delete --exclude '.X*' /opt/public /home/
echo "" > /home/public/.local/share/recently-used.xbel
rm /var/spool/cups/*
```

My lightdm.conf:

```
[Seat:*]
autologin-guest=false
autologin-user=public
autologin-user-timeout=5

session-cleanup-script=/usr/local/bin/restore
```

Other things:

- I install Google Chrome since pretty much everyone is familiar with it regardless of what system they're used to running. The keychain password is set to blank/unencrypted.
- I change LibreOffice Writer to save to docx as default.
- I add shortcuts to Chrome and Writer to the desktop.
- I remove terminal from the quick launch toolbar since most users don't know what it is.
- I remove logout and shutdown options from the start menu. (This is a PITA to do by hand, but here it goes:

In `/usr/share/cinnamon/applets/menu@cinnamon.org/applet.js`, find the line `//Lock screen`. Start a multiline comment there with `\*` and go to the line `//Shutdown button`. Somewhat close to that one, 15 lines or so, there should be a line saying `this.systemButtonsBox.add(button.aactor, { y_align: St.Align.END, y_fill: false });`. End your multiline comment after this line with `*/`.

Finally, I set my preferred volume and remove the volume icon from the toolbar. All our clients have headphones, and they can leak quite a lot if the volume is high enough.

5.3. Configuration options

The Libki client may be customized further with options set in the configuration INI file that are not revealed in the installer wizard.

5.3.1. Labels

To modify the labels on the Libki login screen, you can set a `[label]` block. There valid options are:

- username
- password
- waiting_holds
- error codes received from the server

You need only add lines to the `[label]` block for the labels you wish to modify. Any labels not redefined here will use the default word or words for the given language in use.

```
[labels]
username="Username:"                        ; What text it will say at the username
input field
password="Password:"                        ; What text it will say at the password
input field
waiting_holds="You have holds waiting."     ; This can be used if your client connects
to your library's ILS via SIP2.
INVALID_USER="Login Failed: Username and password do not match" ; What text to show on
the login screen when this
                                            ; error code is received from the server
(any error code can be used)
```

Labels translation

To assign custom text by languages, you can define a different `[label]` block for each specific supported language. The syntax for the new block is `[labels-<lang>]` where <lang> is the language code with or without the country part. For example : `[labels-en]` or `[labels-fr_CA]`.

```
[labels-fr_CA]
username="Username:"                        ; What text it will say at the username
input field
password="Password:"                        ; What text it will say at the password
input field
```

Labels in `[label]` will be used as default values if no label is defined in the language block or if the block itself doesn't exist.

5.3.2. Passwordless login

If you are using single-signon via SIP, and your SIP server is set to mark any password a users provides as valid, you can set the Libki client to passwordless mode.

To do so, simply add or modify the `no_passwords` key under the `[node]` section to appear like so:

```
no_passwords=1 ; Lets you hide the password field if passwords are not used.
```

5.3.3. Restrict client usage by age

It is possible to specify that a given Libki client can only be used by persons of a given age range. To use this feature, just add a new key under the `[node]` section of the Libki client ini named `age_limit`. This feature only works when using single-signon via SIP, and the SIP server returns a `PB` field of the date format `YYYYMMDD` where that date is the patron's date of birth.

Example:

```
[node]
age_limit="gt18"
```

This will limit the client to patrons older than 18. At the moment this only works via SIP2 as there is currently no way to edit a user's age from the staff administration.

Multiple age limits can be implemented delimited by commans, such as `age_limit="ge11,le17"` which will limit the client to users between (and including) the ages of 11 and 17.

The format first two characters are the comparison. Supported comparisons are:

- `eq`: equal to
- `ne`: not equal to
- `lt`: less than
- `gt`: greater than
- `le`: less than or equal to
- `ge`: greater than or equal to

It is possible to make a client unusable by anyone (e.g. "gt18,lt17") so be careful with this configuration.

5.3.4. Automatic logout due to inactivity

It is possible to configure the Libki client to log a patron out if no mouse movement has been detected in a given number of minutes. Additionaly, the user can be first warned after a different number of minutes. Both of these configurations exist as system settings on the Libki server.

It is also possibly to specifiy these configurations on a per-client basis, in the Libki client INI file. If these settings exist in the client INI file, they will override the settings from the server. In this way, the server settings can act as a default, with particular clients overriding those server provided settings.

Example:

```
[node]
inactivityLogout=5
inactivityWarning=3
```

In the above example, Libki will display an inactivity warning after 3 minutes with no mouse movement. The client will then log the user out after another 2 minutes of inactivity. If no warning is wanted, simply set inactivityWarning to a greater number than inactivityLogout.

5.3.5. Run external program on login

It is possible to configure the Libki client to launch another program when a user successfully logs into Libki.

Example:

```
[node]
run_on_login="C:\\Program Files\\Internet Explorer\\iexplore.exe"
```

In the above example, Libki will launch Internet Explorer after the patron has logged in successfully.

You can use single forward slashes instead of the double backslashes (e.g. `C:/Program Files/Internet Explorer/iexplore.exe`)

6. Backup & restoration

It is **HIGHLY** advised to only perform backups and restorations of the database while the system is not being used!

6.1. Backing up the database

The backup program is intended to be used on a server with a libki user.

A backup folder will be created in the libki user's home directory, where the backups are stored.

Please note that this will only create local backups. It is adviced to transfer them to a safe location. A guide on how to transfer automatic backups to an FTP server will be added later.

6.1.1. Make a backup manually

Simply run `libki-backup`.

The backup will be created and stored in `/home/libki/backups`. The file name will contain date and time for the backup for easier identifying, i. e. `db_180706_19.30.sql`.

6.1.2. Make automatic backups

The backup program features a silent mode, perfect for when you don't want any user input or text output. Run `libki-backup -s` to run it in silent mode.

Automatic backups work the same way as the manual backups, but does not provide any output text. The backup file is created directly in `/home/libki/backups`.

If you want to make backups automatically according to a schedule, this is easiest done through crontab. I won't go through how crontab works, but will provide some example schedules.

Start by running `crontab -e`. You will probably want to use the default editor. Add the schedule at the bottom of the file.

Crontab examples

Daily backup at 02:00:

```
0 2 * * * /usr/local/bin/libki-backup -s
```

Backup every Sunday evening:

```
0 21 * * SUN /usr/local/bin/libki-backup -s
```

Backup the first date in every month at 01.15:

```
15 1 1 * * /usr/local/bin/libki-backup -s
```

6.2. Backing up the server files

The easiest way to backup the server files is to tar the libki-server folder in the libki user's home folder. I suggest you put it in the same backup folder as the database backup.

```
tar -pzcvf /home/libki/backups/libki-server-backup_190816.tar.gz /home/libki/libki-
server
```

6.3. Restoring a database backup

Restoring a backup of the database will delete the current database, so be sure you really want to do this.

Start by navigating to the folder containing the backup you want to restore. If the backup was created with the backup program, this folder will be `/home/libki/backups`.

```
cd /home/libki/backups
```

Now run the program and specify what backup you want to restore.

```
libki-restore db_180706_19.30.sql
```

Your database is now at the point it was when the backup was created.

6.4. Restoring a backup of the server files

Start by removing the current server files and replace them with the backup archive.

```
service libki stop
rm -rf /home/libki/libki-server
tar -pzxvf /home/libki/backups/libki-server-backup_190816.tar.gz /home/libki/libki-
server
service libki start
```

7. Updating the server

7.1. Precautions

It is **HIGHLY** advised to only update the server while the system is not being used and the service is shut down.

```
service libki stop
```

Then make a backup of your current server and database, in case something doesn't go as smoothly as planned. Updating isn't entirely stable, especially when updating from very old installations.

Please read through the backup section on how to backup your server.

7.2. Get the latest version and updating the database

Enter your libki-server folder and use `git pull` to download the latest version. Run `installer/update_db.pl` in order to update the database.

```
cd libki-server
git pull
./installer/update_db.pl
```

OBSERVE that if you're upgrading from a 2.x installation, this won't work. You'll need to use the latest 2.x release r19.08 as a stepping stone before updating to a 3.x server, in order to convert the database.

```
cd ~
apt update && apt install unzip -y
wget https://github.com/Libki/libki-server/archive/r19.08.zip
unzip r19.08.zip
./r19.08/installer/update_db.pl
```

Now you can run the latest `update_db.pl` again.

7.3. Updating the server files

Once the database is up to date, it's time to update the server files. Delete the old server files, replace them with the new ones and change ownership of them to the libki user.

```
cd ~
rm -rf /home/libki/libki-server
cp -r libki-server /home/libki/libki-server
chown -R libki:libki /home/libki/libki-server
```

Once all that is done, it's time to start the service again. If everything worked out properly, your Libki server should be updated and working just as well (or even better) than before.

```
service libki start
```

8. Contributing

8.1. Client & server

8.1.1. Submitting bug fixes and features

Formatting patches

- File an Issue on GitHub in the matching repository (client, server, manual, etc...) for the issue (for both enhancement **and** bug fixes)
  - The earlier you file the better. It's much better to file an issue stating your intention to develop a new feature or fix before you start to give other community members a chance to review the issue and give feedback.
- The subject line of your commit(s) should begin with "Issue XX: " followed by the description of what the patch does.
- Include a hyperlink to the GitHub issue this patch resolves.
  - GitHub will notice and automatically link the commit to the issue.

Libki server

Pull requests for features and fixes to the Libki server should be submitting as pull requests via the Libki server GitHub repository.

Libki client

Pull requests for features and fixes to the Libki client should be submitting as pull requests via the Libki client GitHub repository.

8.1.2. Coding guidelines

Server coding guidelines

- Perltidy new or altered pieces of code before submission.
    - Libki server comes with a .perltidyrc file that should always be used when tidying Libki server code.

8.2. Translation

8.2.1. Translating the server

Starting a new translation

The server is translated by a .po file located in lib/Libki/I18N. The .po file is based on a template in the same folder called messages.pot. It is generated dynamically from the server and then combined with a dictionary of our own.

Before doing anything else, we need to add the language we're translating to the dictionary.

```
cd lib/Libki/I18N
nano extras.pot
```

The first part of that dictionary is the important bit. Add your language to the language menu. Follow the same syntax the other languages use and place it alphabetically. Here I'm adding Danish, so that would be on top. A complete list of language codes can be found here.

```
# Language menu

# Danish
msgid "lang.da"
msgstr "Dansk"

# English
msgid "lang.en"
msgstr "English"
```

You will also need to add this to all the other language files. Since these have been written at different times it might be located in the beginning or in the end of the file.

Now let's update messages.pot by generating a fresh one. It should be generated anew whenever a translation is being worked on. To generate a new one, make sure libki-server is your current directory.

```
./script/utilities/translate.sh --generate
```

Copy the template to your language and edit it.

```
cp lib/Libki/I18N/messages.pot da.po
```

It is strongly suggested to use a dedicated po editor. We recommend Poedit.

Updating an existing translation

If you want to update an existing translation you need to generate a new messages.pot and update your translation so it's based on the new messages.pot. As a part of the update process things that have changed will be marked up, making them easy to find.

First, generate a new messages.pot. Make sure libki-server is your current directory.

```
./script/utilities/translate.sh --generate
```

Once it's generated, update the existing translation. This too must be run from libki-server. We're using Danish as our translation here.

```
./script/utilities/translate.sh --update lib/Libki/I18N/da.po
```

A backup will be created in lib/Libki/I18N in case you need it. Just remember to delete it before making a pull request.

If you're using Poedit to edit your translation it's easy to find the next incomplete line with shift-ctrl-down.

8.2.2. Translating the client

The client is translated with a .qm file generated from a .ts file. We're editing the .ts file and once that's done we generate a .qm file. You need to have Qt Linguist installed. On Linux, that's qt5-dev-tools. `sudo apt install qt5-dev-tools`. On Windows, the easiest way is to download it from Thruask's github. https://github.com/thurask/Qt-Linguist/releases

First we need to edit Libki.pro. Open it and add your language to the list of translations. Use the same syntax as the other ones have. Here I'm adding Danish. A complete list of language codes can be found here.

```
TRANSLATIONS = languages/libkiclient_fr.ts \
        languages/libkiclient_sv.ts \
        languages/libkiclient_da.ts
```

Next is the creation of the .ts file. This is done with `lupdate Libki.pro`, either in terminal, cmd or Powershell.

Now use Qt Linguist to translate your .ts file. It's located in the languages directory.

When that's done, create a .qm file. `lrelease Libki.pro` This will also update the other language files as well.

Finally, add the .qm file to libki.qrc. Here I've added Danish at the bottom.

```
...
    <qresource prefix="/">
        <file>languages/libkiclient_fr.qm</file>
        <file>languages/libkiclient_sv.qm</file>
        <file>languages/libkiclient_es.qm</file>
    </qresource>
</RCC>
```

8.3. Manual

Suggestions for edits can be sent to the Libki Documentation Team as a merge request via GitHub.

To join the Libki Documentation Team, please contact Kyle Hall via the Libki developers mailing list or the Libki Slack workspace.

The manual is currently available on GitHub.

9. Credits

9.1. Authors

- Kyle M Hall <kyle@kylehall.info>
- Erik Öhrn <erik.ohrn@gmail.com>
- Christopher Vella <chris@calyx.net.au>
- Luke Fritz <ldfritz@gmail.com>