LibreCores Developer Documentation Documentation

Release 38889e9

The LibreCores Contributors

Nov 14, 2018

Contents

You have arrived at the developer documentation for the LibreCores web site, https://www.librecores.org. This documentation helps you to understand design decisions, the code, and all necessary tools and processes to start contributing.

If you're new to developing LibreCores, the *Quickstart Guide* is a good starting point.

CHAPTER 1

Documentation Contents

1.1 LibreCores Development Environment

To contribute to the LibreCores site, you need to first setup a local development environment. Don't worry, it's done quickly.

1.1.1 Quickstart: Set up a LibreCores development environment

Note: All documentation assumes that you're running Linux. Unless noted otherwise, we have tested Ubuntu 16.04 and openSUSE Tumbleweed. If you're running another operating system or version, please get in touch when you run into trouble. Note that a Linux installation inside a virtual machine (VM) is usually not sufficient as the development environment runs inside a VM itself.

Step 1: Get the code from git

First, you need to get the current development code from Git.

```
cd your_preferred_code_directory
git clone https://github.com/librecores/librecores-web.git
```

Note: Throughout the documentation, most file paths will be given relative to the directory where you cloned the librecores-web repository.

Step 2: Provide secrets

LibreCores communicates with some third-party APIs, such as GitHub or Google, for user logins and other things. To talk to these APIs, personalized API keys ("secrets") are needed.

First, copy the default secrets file as starting point:

```
cp ansible/secrets.dist/dev-vagrant.secrets.yml ansible/secrets/dev-vagrant.secrets.
↪yml
```

Now you have two options:

- Either open the `ansible/secrets/dev-vagrant.secrets.yml` and follow the instructions in there to get a custom API key for the various providers, or
- Leave the file as-is and therefore disable all third-party APIs. This is sufficient if you do not want to change the user-login process.

Step 3: Set up the development environment

LibreCores uses Vagrant and Ansible to provide a development environment that's very similar to our production environments. To simplify the setup of all necessary components, we provide a bootstrap script for Ubuntu and openSUSE. Simply run

```
./bootstrap-dev.sh
```

to install all dependencies and get started.

Alternative version of step 3: Manually installing the development environment

If you're not using openSUSE or Ubuntu, you need to manually install and configure the required dependenies.

- Install VirtualBox, if you don't already have it installed.
- Install Ansible >= 2.0
- Install the NFS server packages. If you don't want to, see the note on NFS below.
- Install Vagrant. It's fast and simple: http://www.vagrantup.com/downloads.html
- Install vagrant-hostmanager: `vagrant plugin install vagrant-hostmanager`
- `cd vagrant; vagrant up`. This might take a while.
- Take your web browser to http://librecores.devel and you should see the LibreCores web site.

A Note on NFS

NFS is used by Vagrant for sharing the development files between your host and the VM. NFS is the most reliable and performant way, but requires a bit more setup.

If you use NFS, . . .

- Install the NFS server on the host machine. Usually the package is named nfs-kernel-server.
- Make sure that the daemon runs (`sudo sytemctl status nfs-server`)
- Disable any firewall rules preventing access from vboxnet devices to the NFS ports. On openSUSE, by default the "ext" firewall rule is applied, preventing access. The easiest way is to disable the firewall completely.

If you don't use NFS, . . .

- Open the file `vagrant/Vagrantfile` and uncomment the corresponding lines of configuration.

Customizing Vagrant Configuration

If you need to customize Vagrant configuration, such as adding networks or you can add a `Vagrantfile.private` under the *vagrant* folder with your customizations. However, not all settings in `Vagrantfile` can be overridden.

Example: The following configuration uses SSHFS in place of NFS (you need to disable NFS in your host though) and attaches the VM to a bridged network to acquire a public IP.

```
Vagrant.configure("2") do |config|
    config.vm.synced_folder ".",  "/vagrant", id: "vagrant-root",type: 'sshfs'
    config.vm.synced_folder "..", "/var/www/lc", id: "application", type: 'sshfs'
    config.vm.define 'librecores' do |node|
        node.vm.network :public_network, ip: '10.42.0.99'
    end
    config.vm.provider :virtualbox do |v|
        v.cpus = 2
        v.memory = 2048
    end
end
```

Step 4: Develop!

Now that all setup is done, you can start developing. First, point your web browser to http://librecores.devel. This will open the development version of LibreCores running on your machine. Whenever you make a code change, a simple reload of the page in your browser is usually sufficient to show the changes.

Note: By default, two test users are created which you can use to log in: user "test" with password "test", and user "test2" with password "test2".

Happy coding!

1.1.2 Tips and Tricks

When developing, some commands and tools have proven helpful. This page lists some (mostly unsorted) tips and tricks that make you more productive.

Note: We use `vm$>` for commands to be executed inside the Vagrant-based development VM, and `host$>` for commands to be executed on the host PC as your normal user.

Connect to the VM

```
# execute this inside your top-level code directory
host$> cd vagrant
host$> vagrant ssh
```

Rebuild ORM (Doctrine) Entity

```
vm$> cd /var/www/lc/site

# for only one class
vm$> ./bin/console doctrine:generate:entities LibrecoresProjectRepoBundle:Project

# for all classes
vm$> ./bin/console doctrine:generate:entities LibrecoresProjectRepoBundle

# finally, update the MySQL DB
vm$> ./bin/console doctrine:schema:update --force
```

Access the MySQL database

Note: In the Vagrant development environment you can connect to the database with user "root" and password "password".

To access the database through a web frontend, phpMyAdmin is your friend. You find it at http://pma.librecores.devel.

If you prefer to access the MySQL database on the command line, you need to SSH into the VM.

```
# use the mysql client to perform queries
vm$> mysql -uroot -ppassword librecores

# use mysqldump to get a dump of the whole database (or parts of it)
mysqldump -uroot -ppassword librecores
```

(Yes, the password is "password".)

Asynchronous Processing with RabbitMQ

Access the RabbitMQ management plugin

http://librecores.devel:15672

- Username: admin
- Password: password

Run the consumer

```
vm$> cd /var/www/lc/site
vm$> ./bin/console rabbitmq:consumer -m 1 update_project_info
```

Test the producer: update one project

```
vm$> cd /var/www/lc/site
# update the project information of project 1 (needs the consumer!)
vm$> echo 1 | ./bin/console rabbitmq:stdin-producer update_project_info
```

Empty the queue

```
vm$> sudo rabbitmqctl purge_queue update-project-info
```

Run the consumer in debug mode

Sometimes the consumer script re-spawns and no useful error messages are available in the log files. In this case you can manually run the consumer to see the log messages.

```
# on staging/production
sudo systemctl stop librecores-rabbitmq.service
SYMFONY_ENV=prod SYMFONY_DEBUG=0 sudo -E -u www-data -- /usr/bin/php /var/www/lc/site/
↪bin/console rabbitmq:consumer -vvv -w -l 256 -m 1 update_project_info
sudo systemctl start librecores-rabbitmq.service
```

Clean the Symfony caches

```
vm$> cd /var/www/lc/site
vm$> ./bin/console cache:clear
```

Remote PHP debugging

The development environment has Xdebug remote debugging enabled using the common default settings: `xdebug.remote_port` is set to port 9000 and *xdebug.remote_connect_back* is set to `1`. Please refer to your IDEs manual for further information how to make use of this functionality.

Check the coding style of PHP code

```
vm$> cd /var/www/lc/site
vm$> ./vendor/bin/phpcs --runtime-set ignore_warnings_on_exit true -s \
  && echo You can commit: No errors found!
```

Use Algolia

LibreCores makes use of Algolia to provide the search functionality. Some settings of Algolia can be managed through its web UI, but most data and configuration is pushed from the LibreCores server. In a development environment using Algolia is optional (if it is not used, no search functionality is available). If Algolia should be used, first register an account at their web page (the basic account is free and sufficient for development).

Then the configuration needs to be inserted into the LibreCores web app (all data is available from the Algolia web UI). Specify the application id (`site_algolia_app_id`), the admin API key (`site_algolia_api_key`) and the search API key (`site_algolia_search_api_key`) in the corresponding configuration file in `ansible/secrets` (use `dev-vagrant.secrets.yml` for the development settings in Vagrant).

Then push the configuration to Algolia using the `search:settings:push` command (see below).

Afterwards push the data to the search indices using `search:clear` followed by `search:import`.

Clear indices

The data stored with Algolia can be removed using the following commands.

```
vm$> cd /var/www/lc/site
vm$> ./bin/console search:clear
```

Push data to Algolia (indexing)

To send data to Algolia to index it the data needs to be "imported". This can be done using the following commands.

```
vm$> cd /var/www/lc/site
vm$> ./bin/console search:import
```

Backup settings

```
vm$> cd /var/www/lc/site
vm$> ./bin/console search:settings:backup
```

Push settings to Algolia

```
vm$> cd /var/www/lc/site
vm$> ./bin/console search:settings:push
```

1.2 Contributing to LibreCores

LibreCores is a community-driven project, we are always looking for people to contribute to LibreCores: to evolve existing features, to try out new ideas, to fix bugs from small to large. Many contribution opportunities do not even require coding skills: writing documentation, improving the design, testing and giving feedback is just as valuable as code commits are.

Are you ready to help out? We're put together a couple of ways for you to get started, depending on the amount of time you have available, and the skills you have.

In any case, get in contact if you have any question and remember, no question is too dumb to be asked. All of us started small at some point in time. We have a couple options for you got get in contact with us.

- **Chat with us on Gitter**
  - Join the chat at http://gitter.im/librecores/Lobby. Most of the devs hang out there.
  - If you don't get an immediate answer just wait, most of us are based in European time zones. We try to get back to your question as soon as we see it.
- **Write an e-mail to the mailing list dev@lists.librecores.org**
  - You need to sign up for the mailing list first here: https://lists.librecores.org/listinfo/dev
  - After you've signed up, you can write a mail to dev@lists.librecores.org and will receive all replies.
- **Talk to a human**

  - if you have trouble with the means of communication above contact Philipp (@imphil on IRC and GitHub) by dropping him a mail at mail@philipp-wagner.com.

1.2.1 Work on a "Good First Bug"

- Time required: varies, from a couple minutes to a couple of days.
- Skills required: varies, see the issue report for details. Usually ranges from from testing to graphics design to coding.

In our issue tracker on GitHub, we have labeled some bugs as "good first bug." These bugs contain a clear description what needs to be done, a description of skills which are required, and a mentor who's available to help out.

List of Good First Bugs on GitHub

If you have any questions, please don't hessitate to contact the listed mentor, or comment in the GitHub issue. We understand that it's hard to get started at times, so if you run into trouble setting up the development environment, don't understand anything in the project description, or just want to get feedback if your approach works, please let us know.

1.2.2 Testing and QA

- Time required: usually not more than one hour
- Skills required: varies, see the issue report for details.

Every user has a slightly different setup when using LibreCores. To make sure the site works flawless for as many users as possible, testing and quality assurance (QA) is important – and it's a great way to contribute to LibreCores! If you see any problem with the LibreCores site, please open an *issue report on GitHub <https://github.com/librecores/librecores-web/issues/new>*. If you're looking for ideas what to test, we label some issues with the "qa wanted" label. These issues usually contain a description of what changed, and what should be tested to make sure it works. If not, please ask the author of the bug.

List of QA Wanted Bugs on GitHub

1.2.3 Creating documentation, reports and tutorials

- Time required: from hours to days
- Skills required: English writing skills (don't worry if you're not a native speaker!)

LibreCores is not just a project repository, but also a community hub for people to get started with digital hardware design. We try to answer questions, such as "What license should I choose?", "How do I setup a development environment on my PC?", or "What's the process to create an FPGA design"? Creating such content, in the form of tutorials, reports, documentations, videos or even interactive forms like games, is a great opportunity to contribute to LibreCores. No coding skills are required, and if you feel that your English is not sufficiently good, don't worry either! Most of us are not native speakers, and we'll find someone for proof reading if you wish!

If you already have an idea what to write, just open a text editor, Word, and type ahead! We'll take care to get it formatted properly for the site when it's time for it.

If you need some inspiration, have a look at the issues in our GitHub issue tracker labeled as "content".

1.2.4 Take on a larger project

- Time required: from days to weeks
- Skills required: usually coding (mostly PHP/Symfony) skills are required

If you're ready to spend it a bit more time on LibreCores, we've got something interesting for you as well. While not all our ideas for improvement have been written down, some have. Look at the "userstory" or "enhancement" labels on our GitHub issues for some inspiration. If you have another idea, just create a new issue or jump on to the development mailing list.

1.3 Coding Style

A common coding style helps to keep our code readable and maintainable. Towards that goal the most important rule is: write code in the same style as existing code. Look at current code and try to mimic what you see. If you have good reason to do things differently, do so.

Find below a discussion of the coding style we use for the different programming languages.

1.3.1 PHP

The LibreCores PHP code follows mostly the Symfony Coding Standards. The following exceptions/changes and extensions are made:

- Try to keep lines at 80 characters length.
- Yoda-style comparisons (e.g. `if (0 === $someVar)`) are discouraged.
- Use the `||` and `&&` operators for logical AND and OR (instead of `and` and `or`).
- Start inline comments with a single space, e.g. `//  comment`).

The PHP code can be checked using the phpcs tool. The required configuration is part of the source repository (`site/phpcs.xml.dist`) and will be picked up by phpcs automatically.

1.4 LibreCores Design Documents

Some design decisions taken by the LibreCores web site have been the result of a long thought process, or a lengthy discussion. To better understand *why* code has been written in a certain way, the LibreCores design documents explain the reasoning and help when revisiting the designs.

1.4.1 LibreCores User Management

Account Creation

Local Account

Local account creation follows the usual procedure, known from online services.

Prerequisite: The user is not logged in on the LibreCores site.

1. The new user enters username, email address and password
2. Account checking

1. If username and email address are not in the database yet, a new user account is created.
2. If username or email address already exist in the database, the new user nees to pick other values, or log in with his/her existing account.

Final state: The user is either logged in with the new account (account creation successful); or remains not logged in (account creation failed).

OAuth Account (GitHub/Google)

OAuth-based account creation avoids that the user needs to enter his/her user data again if he/she has already registered with a OAuth account provider (currently supported: GitHub and Google). In addition, the user can log in with through the OAuth provider, instead of entering the user credentials on the LibreCores site.

Prerequisite: The user is not logged in on the LibreCores site.

1. The new user clicks on the OAuth provider logo.
2. The new user is redirected to the OAuth provider, where he/she logs in and/or confirms the data transmission to LibreCores. The exact steps depend on the OAuth provider itself.
3. The user is redirected back from the OAuth provider to the LibreCores site with a OAuth token.
4. The LibreCores site uses the received OAuth token to request user data from the OAuth provider. Requested user data: email address and the username used on the OAuth service.
5. If the OAuth response does not contain an username or an email address, the account creation fails with an error message.
6. The LibreCores user database is searched for an existing account associated with the username or the email address.
7. The OAuth provided username and email address are used to create a new user account.
8. User validation

1. If the newly created user does not validate, e.g. because the username provided by the OAuth provider does not follow our rules or is already taken, the user is redirected to the registration form. The registration form is pre-filled with the data used for the automatic account creation and left for the user to modify. After submitting the registration form successfully, the newly created account is automatically connected to the OAuth account provided in the first place.
2. If no account with either the email address or the username exists, a new LibreCores user account is created. The email address and the username are filled with the data from the OAuth provider. The username received from the OAuth provider, together with the OAuth access token, are stored in the user object.

Final state: The user is either logged in with the new account (account creation successful); or remains not logged in (account creation failed).

Login

Local Login

The username or e-mail address is checked against the local database.

OAuth Login

A button “Login with SERVICE” is provided, which redirects the user to the auth provider. After a successful login, the auth provider redirects the user back to the previously viewed page on LibreCores. If no user account is associated with the OAuth account, the account is automatically created (see “Account Creation” above).

Connecting an Existing Account with an OAuth Provider

Users can connect (associate) an existing account on LibreCores with an OAuth-provider. Multiple OAuth accounts can be connected to a single LibreCores account. Every OAuth account can be connected only to one LibreCores account.