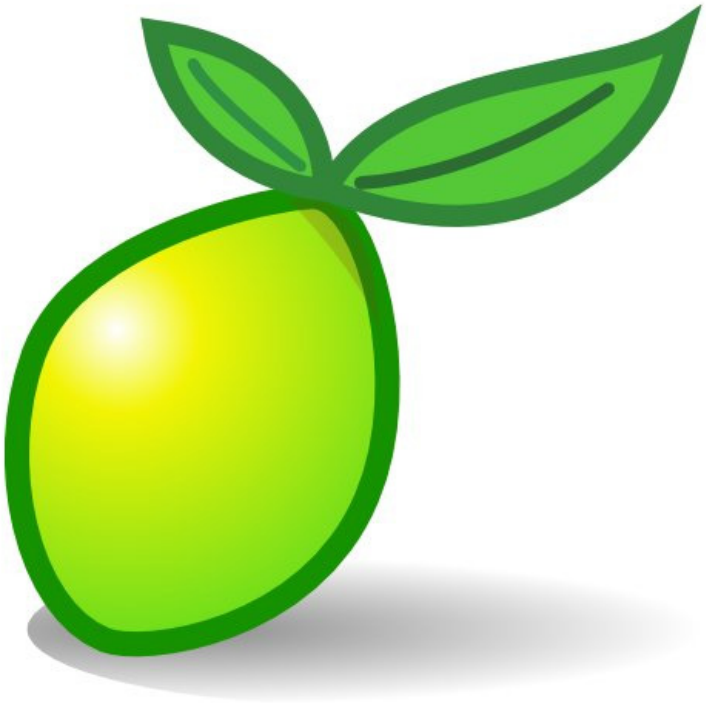


LIMESURVEY 2.0

'JUMPING ABOARD'

Preface

Joining an open source software project can be fun but isn't always easy. There's a lot of code to go through, you are getting to know the team members, then the coding rules, setting up your work environment… getting started even before you code becomes an unnervingly complex experience.

We produced this handbook as a step away from the prevailing *'Just dive in'* mentality in the world of open source. It is not intended to be extremely detailed; it's not there to hold your hand or dry your eyes when things get difficult. But it should give you important pointers and get you prepared for the wilderness ahead.

For whom is this handbook

This handbook is assuming that you, the reader

- are a seasoned programmer
- have experience with PHP and web-development in general
- have experience with JavaScript and esp. the jQuery framework
- are able to set up and configure a web server yourself on your own computer
- understand Relational Databases and have worked with SQL
- know how to use SVN or at least have used a different version control programs
- are willing to learn new things
- are self-motivated
- want to help LimeSurvey ☺

What if I'm not really that interested in surveys?

You don't have to! LimeSurvey 2.0 is in fact conceptualized as **a tool for general purpose collecting and analysis of data.** Surveys are one such use, but it can be equally used for making e.g.:

- quizzes
- online tests
- sign-ups forms (e.g. for volunteers)
- forms for data entry

So, are you ready? Then let's dive in!

Getting ready

First of all, let's introduce the project itself: LimeSurvey 2.0 is a **web-based tool**, developed and targeted for **PHP 5.2.0+**, with **support for most popular database systems** (incl. MySQL, Microsoft SQL, Oracle, Postgres), thanks to the **CakePHP framework**, upon which it is built. The interface uses lots of JavaScript, leveraging **jQuery** and many of its plugins. Finally, the project source code is hosted on **SourceForge**, with access and version control maintained by **SVN**.

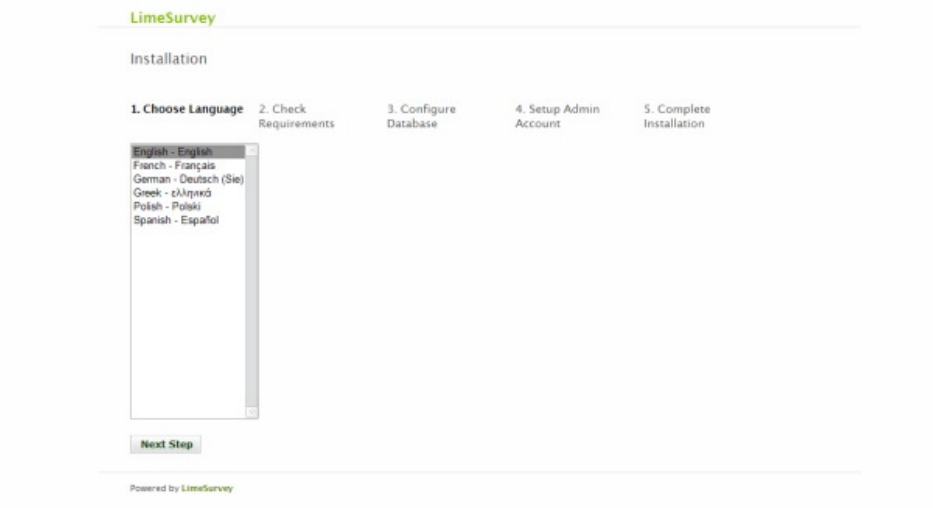
Grab the source code

01 Using the SVN tool of your choice, checkout the source code from this URL: `https://limesurvey.svn.sourceforge.net/svnroot/limesurvey/source/limesurvey20/limesurvey-dev`

02 The checkout will take about 5 minutes, so go fetch yourself a cup of coffee.

03 Once you are done, set up your web server to access this directory, which we will refer to as `<root>`. You can go for any URL scheme you want, so you can either access it as `http://localhost` or `http://localhost/limesurvey` etc.

04 Check that your setup is successful and that you see this in your web browser. Don't run the installation just yet!



05 As a developer, you'll need to see all the errors that are hidden in the production version. It's time to activate the debug mode. Head to `<root>/app/config` and copy `debug-core.php.sample` to `debug-core.php`.

Debug-core.php is a file where you can override the default **core.php** settings. Customize it to your needs but do not commit it!

You must never edit core.php!

06 Go back to your browser, refresh it and install LimeSurvey. Further instructions are at http://bit.ly/lime2install

Important Links

- http://www.limesurvey.org/ - our homepage, check here for news, tweets, links
- http://docs.limesurvey.org/ - wiki pages, acts not only as manual for users but also as discussion ground for developers. Information goldmine, both in the sense of richness as well as requiring you to do a little bit of digging to find what you need.
- http://bugs.limesurvey.org/ - bugtracker for both project. Be sure to tune in to 'Bug reports v2.x' and 'Development Limesurvey 2'.
- http://ideas.limesurvey.org/ - idea 'torrent', discuss the next features and listen to what users have to say
- http://book.cakephp.org/ - the official CakePHP online book, good starting point
- http://jquery.com/ - amazing cross-browser JS framework
- And last but not least, http://www.google.com/ - remember, it knows everything!

Know the team

We hang out on the IRC server Freenode, channel **#limesurvey**. Once inducted as a developer, you'll get access to the developer's channel **#limesurvey-dev** where (unsurprisingly) development is discussed. This channel is also logged (at http://www.limesurvey.org/en/irc-logs-usermenu-114, although you need access from administrators) for later reference, so any important discussion should be carried out there.

We meet up **every Tuesday at 20:00 GMT** so if you have a big topic to discuss, that would be the time.

And here are team member introduction, as written by themselves (in alphabetical order):

- **c_schmitz** – Carsten Schmit, LimeSurvey project lead - contact him on organization issues and any questions you have regarding LS1 and partially LS2
- **El-Matador-69** –
- **eric_t_cruiser** –
- **jasebo** –
- **lemeur** – LS1 code developer and maintainer. Answers questions regarding how messy is LS1 and what NOT to do in LS2
- **macduy** – LS2 maintainer, GSOC 2009 Participant
- **Mazi** - they guy to ask if you have general questions about Limesurvey and features in Limesurvey 1
- **mdekker** –
- **tpartner** – Tony Partner, LS1 developer and user support

Code Overview of LimeSurvey 2.0

Note: LimeSurvey 2 is developed on top of CakePHP. It is assumed you can familiarize yourself with this framework on your own. For more details visit http://bakery.cakephp.org/

Current code status can be found here: <<Missing link>>

The construction of LimeSurvey 2.0 can be thought of as several interacting but otherwise fairly separate components, each concentrating on a particular aspect of the task at hand. This loose coupling helps the code to be maintainable and easier to start working with. You can work on a particular component without disrupting, or even having an in-depth understanding of, the functionality of the other.

Installer

Pretty self-explanatory, but let's have a look at what it does from developer's perspective:

- it creates a **database.php** file in `app/config`. When this file exists, LimeSurvey assumes that installation is completed and will proceed with normal behaviour
- if **Fancy URLs** (i.e. removes `index.php/` from URLs) is enabled, `.htaccess` files, which were previously empty, will be filled. These files are located in `<root>`, `app` and `app/webroot`
- **install-core.php** inside `app/config` is changed. This is a file that is used to override settings of core.php (which is a file that must never be changed). Bit like debug-core.php, but for normal installation.

Management

Management concerns itself with 'things around the actual surveys' – i.e. users, groups, security, the deployment and sending of surveys to the participants etc. It knows nothing of how surveys are built or how data is stored.

Users and Groups

A user is an *agent* interacting with LimeSurvey - he may be an admin, a person who designs surveys, or someone who takes one, or any combination of the above. A User is a member of exactly one Group (more precisely referred to as 'Security Group'). As of now, a Group may not be a member of another Group, but since LimeSurvey uses ACL-based permissions, it would be well worth it to implement this feature.

Surveys

From the management point-of-view, a Survey is just a generic entity. In this section, we do not consider the intricate details of a Survey or what it is composed of.

Each survey has one User as its owner, nonetheless different Users may edit a Survey, through given ACL permissions. A Survey in simplest terms can be considered as a "form" or a "collection of questions" and is on its own **not 'runnable'**. In order to "take a survey", you need to create an 'assignment'.

Assignments

Probably the most confusing part of LS2

In order for a Survey to be used (or "deployed"), it must have an Assignment – something that makes it available. Assignments are a way of reusing the same survey to produce separate sets of data. It is possible to administer the same Survey to two different groups of people, say to the general public, and then to a closed group of selected users. This would correspond to two different assignments. The data collected from both can be analyzed separately, yet they still come from the same Survey.

Non-public Assignments will need to be associated with users. However, it is sometimes the case that a certain group of people will always be given the same surveys (say, administering a series of questionnaires to all the school classes within a grade) and it would be tedious to add them every time to a new Assignment. Hence, Assignment Groups exist, which allow connections between a group of users and several Assignments. A User may belong to several Assignment Groups and Assignment Groups can be linked to several Assignments. Assignments equally can have as many Assignment Groups and as many individually assigned users.

Announcements

Announcements are a way of communicating with users. There can be several types of announcements (currently only one type – e-mail – is implemented), modelled as Announcement Type. Announcements are currently used to notify users that they have an Assignment to do, and will usually consist of a template message and a unique URL to the Assignment.

Announcements are completely optional and a user can complete an Assignment without having to receive an Announcement. He may however receive several announcements, say a first invitation and then several reminders afterwards, because he hasn't completed it. Issued announcements are tracked via the AssignmentMessage model, which contains information about which user it was sent to, what assignment it was referring to, and what Announcement was used.

Coding Guidelines

We appreciate the diversity of coding experience & styles and grant you a "freedom of expression" while writing code. However, to prevent the source files from becoming too much of hodgepodge, we kindly ask you to observe these few rules.

Formatting

Formatting rules are always boring and rather than embarking on a discussion or providing a rationale, just make sure you know and follow them.

- For indentation, we use 4 spaces. **No tabs**.
- Put **1 space** on both sides of arithmetic operators and outermost brackets, unless are a part of function calls. Put **1 space** after commas in arguments. This rule is actually hard to express explicitly, so just follow these examples or follow the code ...

```
INCORRECT:   if(($e+2)==2){
CORRECT:     if (($e + 2) == 2) {

INCORRECT:   $this->add ('line',3,$callback);
CORRECT:     $this->add('line', 3, $callback);
```

- Start each block with a brace on a new line

```
INCORRECT:   if (1 == 1) {
                 do_stuff();
             } else {
                 do_other_stuff();
             }

CORRECT:     if (1 == 1)
             {
                 do_stuff();
             } else
             {
                 do_other_stuff();
             }
```

- Write inline arrays, that may have many keys as follows:

```
$data = $this->find('all', array(
    'limit' => 10,
    'page' => 2,
    'conditions' => array(
        'User.id' => 1,
        'User.username' => $username
    ),
));
```

- It is acceptable to write shorter arrays in one line

```
$this->find('all', array('limit' => 10, 'page' => 2));
```

Coding philosophy

Clarity First

Remember, the formatting guidelines are just *guidelines*, they are not hard and fast rules that you must absolutely obey. It is acceptable to deviate from them, as long as you observe this Top Priority Rule, which must be respected above all:

Clarity First!

No amount of fancy formatting can make a code understandable and easy to read as **appropriately named variables** and **clear, explanatory comments**. **Don't underestimate the power or importance of comments** – it may be clear to you what the 50 lines you just wrote mean, but not to the person who might review or fix it after you are long gone.

Acceptable deviation from Formatting guidelines

Popular among experienced coders is to smack *if*s into a single line. This is acceptable if the code in question does some mundane, or otherwise uninteresting operation that can be explained using an accompanying one-line comment. Instead of 6-lines, write

```
// Select a source array if it has been passed, otherwise use $_GET
if (empty($data)) $source =& $data; else $source &= $_GET;
```

Variable naming

We don't force you to use prefix notation. Our philosophy is, rather than naming a variable something like *$iaIDs*, which supposes the reader to know that *i* means *integer* and *a* means array, feel free to name it as appropriately, possibly including a short comment at declaration. We'd much more appreciate:

```
$ids = array();     // array of User IDs
```

Write documentation properly

Document every function you create, describing what **it does** and what are **the options/parameters**. Give usage examples if necessary. If the function is too obscure or is a helper function, which serves one purpose only (perhaps to break up the code into manageable pieces), just describe why is it there and **where it is used/called.**

```
/**
 * Formats the data received from a find() function into an array, that can be
 * later converted to JSON
 *
 * @param $data     Data returned from a find() function
 * @param $callback Callback function, which returns an array of data for each
 *                  element of $data passed to it
 * @return An array, ready for conversion to JSON
 * @example jqData($this->User->find('all'), '__callback_function');
 * @author macduy
 */
function jqData($data, $callback) …
```

Don't start a documentation with 'This function does' or 'This variable is used for'. Be short and concise, go straight to the point!

Own up! Use @author!

Always use @author to denote functions or modules you have written or amended significantly. **This is not an issue of modesty, but one of responsibility**. When somebody later wants to upgrade or otherwise completely rewrite a specific function, they may want to consult its original author. Not using @author will be considered bailing on your responsibility – and hopefully it will motivate you to write better code as well.

Try not to repeat yourself

Try to minimize code duplication. Once you delve into development, you'll sometimes find it hard to merge two very similar functions into one, occasionally at the price of having to alter its interface or parameters. Try as hard to come up with a solution, e.g. writing a generalized private helper function and rewrite the original two functions to make use of it.

Use of Language Features

PHP is a modern dynamic two-paradigm language and comes with a plethora of features, some of which we encourage, some are potentially risky and some are outright dangerous.

Think of Sustainability and Manageability

When using some of the advanced PHP features, think of how sustainable and manageable the code will be:

- will others, including me, be able to tell what is going on?
- can I change my code without breaking its usages across the project?

As an example, PHP permits calling a function, whose name is stored in a variable:

```
$method = 'habtmDelete'
$this->{$method}('User', $group_ids);     // $this->habtmDelete will be called
```

Though this approach is abundant in CakePHP, it is extremely rare in LS2. This is because CakePHP is a **framework**, and we rarely need to concern ourselves with the code specifics of it, nor do we need to alter it (in fact, **we ban it!** If you change CakePHP code, and then we later upgrade it, hell will break loose!). The above would be allowed if the function using it is a fairly low-level operation and its purpose and usage well documented.

The Golden Rules

- **Clarity First!** That means: sensible variables and comments comments comments!
- **Don't repeat yourself.** Generalize wherever possible.
- **Don't use PHP features that are too fancy** at the expense of clarity, sustainability and manageability
- **Don't alter CakePHP files!** We regularly update to the latest stable version and obviously your changes.
- If you need to change some core behaviour of CakePHP, use **inheritance** to override the method concerned (e.g. in *app_model.php*, the model's *find()* function is overridden). However, be extremely cautious! **Think of the consequences before having to resort to such a move!**

On your mark, get set, go!

Now that you know how to setup your developer version of LS2 and "know the ropes", take your first steps towards contributing.

Establishing your online identity

Before you start your 'career' as an open-source developer with us, make sure to set up your online identity, i.e.

- **sign up at http://www.limesurvey.org/**. You'll get access to the irc developer channel log, the forum, the wiki docs, bug tracker and idea pool.
- **have an account at http://sourceforge.net/**. We do not give SVN write access straight away and although there is no official selection process, this permission is usually granted after a couple of reviewed submitted patches.
- **Sign up** for the **developer mailing list**
- Register a nick and **say 'hi'** on the IRC channel **#limesurvey** on irc.freenode.org.
- **Make yourself heard and known to us!**

Development process

After you have **planned** and/or, if needed, **consulted** your project with team members, you're ready to code. When coding, please try **to follow the coding guidelines** outlined later in this document.

Make sure that apart from code, you also leave behind some sort of **documentation**, either in form of **comments**, or as a **wiki page** if more space is required. We know that programmers don't always get excited about this, but if you can, diagrams can be very efficient in illustrating an idea or concept.

You can use the bug-tracker and idea torrent as a form of to-do list or a development notebook. Use clear titles for bug reports as others may be interested in them. "Survey Engine fails to run a survey first time" is probably better than "Survey Engine doesn't work" .

If you have fixed a bug or implemented a requested feature, make sure to **make a note/post a reply** on the relevant forum threads, bug reports and similar. **Attract and engage the original audience**, maybe they can review the changes for you!

Remember to keep in touch with your fellow developers to avoid stepping on their feet (or being stepped on!), but also with the users. Listen to their suggestions and calm they down when they start sporting unrealistic suggestions.

Committing to SVN

Some points to note:

- do **not create branches**
- observe the following standard for commit messages
- if your commit refers to a bug report/reports, make sure to **post a note on the bugtracker** in the relevant report. Something along the lines of "Fix committed to r8430" followed by a brief description if necessary, will do.
- if you're submitting somebody else's patch (always only after having reviewed it!), attribute credit to them, e.g.: "Fixed bug 01234: X caused Y to fail (thanks to john_q)"

Current code status

It is important to keep yourselves and other team members up-to-date with what's going on. Make sure to check and update the current code status wiki page from time to time.

<<Missing link>>

Also check out the wiki: http://docs.limesurvey.org/LimeSurvey+Development and open bugs in the bug tracker - http://bugs.limesurvey.org

Closing notes

Hopefully, you've found this handbook useful and now feel ready to lay some code down. Remember, the best way to learn is to try, make mistakes, learn from them, rinse and repeat. On your mark, get set, go! Code away!