Largo Project Documentation

Release 0.5.5

Institute for Nonprofit News

August 10, 2017

Contents

Largo is a responsive WordPress starter/parent theme built specifically with the needs of news publishers in mind and is built and maintained by the news apps and technology team at the Institute for Nonprofit News (INN).

Questions? Comments? Send us an email at largo@inn.org

Project Repository: https://github.com/INN/Largo/

Technical Issues (Bugs, Feature Requests, etc.): https://github.com/INN/Largo/issues

Largo User Guides for Administrators, Authors, and Editors: https://largo.inn.org/guides/

Help Desk (Support Requests): http://support.largoproject.org

Knowledge Base: http://support.largoproject.org/support/solutions

User Forum: http://support.largoproject.org/support/discussions

Twitter: http://twitter.com/largoproject

Mailing List: http://eepurl.com/yu0bT

For Developers

For Developers

Overview

Project Largo is released as open source software and anyone is welcome to download or fork the project from our github repository and use it as they like.

If you use Largo for a project we'd love to hear from you so can we add you to our list of sites using Largo and help get the word out. Send us an email! largo@inn.org

The preferred way of building a site with Largo is by creating a WordPress child theme. We have created a sample, heavily documented, child theme to help you understand the way we structure our child themes in the hopes that it will give you a solid framework to get started. There is more information on setting up Largo and using child themes in the download and installation section of our documentation.

Setting up a development environment

We use a set of tools to make setting up a Largo development environment as easy and consistent as possible.

We encourage all INN member organizations looking to add features to or otherwise modify their theme to use this same setup, since doing so will make support and collaboration between members and INN easier.

Read:

Setting up a complete Largo dev environment

This recipe will walk you through setting up a fresh WordPress install on a Vagrant Virtualbox machine with INN's deploy tools and Largo installed.

We'll walk you through the overall setup of the WordPress directory, and then we'll walk you through setting up Largo and its development requirements.

Software to install first

From INN's computer setup guide, install the following software:

- git
- wget

- curl
- phpunit
- virtualenv and virtualenvwrapper
- an SSH key
- VirtualBox
- Vagrant
- npm and grunt-cli
- xgettext (only needed for rebuilding translation files and releasing)

If you're on OSX, you will also want to install Homebrew, to assist in the installation of the above.

Once you have all that set up, you're ready to install Largo and WordPress inside a virtual machine!

Setting up Largo and WordPress

1. First, create a new directory. This will be version-controlled with git, to make it easier to update Largo and the deploy tools. If you're hosting WordPress someplace that allows SFTP access, our deploy tools will help you version-control your deploy, for fun and profit.

   ```
   mkdir umbrella
   cd umbrella
   git init
   ```

2. Add the deploy-tools and Largo repositories.

   ```
   git submodule add https://github.com/INN/deploy-tools.git tools
   git submodule add https://github.com/INN/Largo.git wp-content/themes/largo-dev
   ```

3. Update all the submodules

   ```
   git submodule update --init --recursive
   ```

4. While you're at it, copy some configuration files from the deploy-toools folders to the project root:

   ```
   cp -r tools/examples/ .
   ```

5. Now, on to Largo.

   ```
   cd wp-content/themes/largo-dev
   ```

6. You're going to have to install some things first.
7. First, install the Python dependencies.

   We use a few Python libraries for this project, including Fabric which powers the INN deploy-tools to elegantly run many common but complex tasks. In the OS X setup guide, you should have installed Python virtualenv and virtualenvwrapper.

   Make sure you tell virtualenvwrapper where the umbrella is.

   ```
   export WORKON_HOME=~/largo-umbrella
   mkdir -p $WORKON_HOME
   source /usr/local/bin/virtualenvwrapper.sh
   ```

You should add that last line to your .bashrc or your .bash_profile.

Now we can create a virtual environment and install the required Python libraries:

```
mkvirtualenv largo-umbrella --no-site-packages
workon largo-umbrella
pip install -r requirements.txt
```

8. Now, the NodeJS dependencies.

   If this command fails, make sure you're in the `largo-dev` directory.

   ```
   npm install
   ```

9. Our API docs/function reference uses doxphp to generate documentation based on the comments embedded in Largo's source code. You'll need to install doxphp to generate API docs.

   - Installation process with PEAR:

     ```
     pear channel-discover pear.avalanche123.com
     pear install avalanche123/doxphp-beta
     ```

   - Installation process with git. This requires you to know where your `bin` directory is

     ```
     git clone https://github.com/avalanche123/doxphp.git
     cd doxphp/bin
     mv doxph* /path/to/bin/
     ```

   The last step may require you to use sudo.

10. Make sure that you have the necessary prerequisites for these next steps.

    From the top level of the project, run the setup routine as described in the deploy-tools documentation

    ```
    fab wp.verify_prerequisites
    ```

    If the script returns any errors, it will also include (hopefully) helpful information on how to rectify the problems. You may need to reinstall curl.

    If the above command fails to run, make sure you have run the `workon` and `pip install` commands listed above in step 7.

11. Time to install WordPress.

    INN's deploy tools have a handy utility that will install any tagged release of WordPress for you.

    (a) `cd` to the top level of your project, on the same level as the tools/ directory where INN/deploy-tools was installed.

    (b) Find the version number of the latest release of WordPress and use its number in the folloiwing command

    ```
    fab wp.install:4.2.2
    ```

    (c) In the computer setup section above, you installed Vagrant. Now, create the virtual machine:

    ```
    vagrant up
    ```

    (a) While you're waiting, why not stand up, stretch, and make a cup of tea? Downloading the virtual machine disk image and provisioning it will take a while.In that time, it downloads the image of a Ubuntu Linux system, installs the MySQL and PHP servers, along with all of the most recent updates, and configures it just so that all the Fabric commands work.

(b) When it's done, edit your */etc/hosts* file:

```
sudo nano /etc/hosts
```

Enter your password, use the arrow keys to position the cursor at the end of the file and add the following line:

192.168.33.10 vagrant.dev

Then use Ctrl-O to save your changes and Ctrl-X to exit the editor.

This tells your system that whenever you use the address `http://vagrant.dev`, you really mean the IP address of the virtual machine. If you're working on a multisite instance of WordPress, you can add the subdomains such as `another.blog.at.vagrant.dev` at the end of the line, separated by a space from `vagrant.dev`.

5. Now that the vagrant box is up and running, you can create a database for it to use:

   Without any arguments, this command will read the defaults from the `Fabfile.py` in the root of your project directory.

   ```
   fab vagrant.create_db
   ```

6. Now, let's take a snapshot of the virtual machine in its new, provisioned, freshly-deployed state.

   You can name the snapshot anything you want, and I would recommend describing it in a short way that describes what that state would give you if you were to revert.

   ```
   vagrant plugin install vagrant-vbox-snapshot
   vagrant snapshot take default snapshot_name_goes_here
   ```

7. Now you're going to set up WordPress on Vagrant. Open a browser and point it at http://vagrant.dev/. You should automatically be redirected to http://vagrant.dev/wp-admin/setup-config.php. Choose your language, then enter the details below as they are entered in your `Fabfile.py`:

   ```
   * Database Name: `largoproject`
   * User Name: `root`
   * Password: `root`
   * Database Host: `localhost`
   * Table Prefix: `wp_`
   ```

8. If you are working on a multisite install, you will want to add these settings to `wp-config.php` at the bottom, before "Do not edit below this line."

   ```
   /* Make this a multisite install. */
   define('MULTISITE', true);
   define('SUBDOMAIN_INSTALL', true);
   define('DOMAIN_CURRENT_SITE', 'vagrant.dev');
   define('PATH_CURRENT_SITE', '/');
   define('SITE_ID_CURRENT_SITE', 1);
   define('BLOG_ID_CURRENT_SITE', 1);
   ```

   All done? Log into WordPress and start poking around. Remember to take Vagrant snapshots when you get things working how you like the. You'll probably want to take one after you add some posts and configure your menus for testing purposes. If you want to log into the vagrant box, it's as easy as `vagrant ssh`.

You have installed:

- INN's deploy tools
- the Largo theme

- Grunt and the nodejs packages we use to handle a bunch of things
- pip, virtualenv, a largo-docs virtualenv, sphinx, and everything needed to rebuild the documentation
- doxphp and dpxphp2sphinx
- WordPress on a Vagrant virtual machine

Some notes about Vagrant

You can work on files without booting Vagrant, but if you want to view the effects of changing the files, you'll want to run `vagrant up` from the root folder of your project, the one that contains the `Vagrantfile`.

If you want to turn vagrant off for a while, run `vagrant suspend`. Suspended vagrant boxes can be brough back to life with `vagrant up`.

When you want to shut down Vagrant, run `vagrant halt`.

If you want to poke around in the Vagrant box, run `vagrant ssh`. You don't have to enter any passwords or unlock any ssh keys - Vagrant controls those itself.

If you're unable to log in, try powering the Vagrant machine off through the Virtualbox graphical user interface, or by finding the VM name in `VBoxManage list runningvms` and using it in `VBoxManage controlvm <name|uuid> acpipowerbutton`

Some notes about deploy-tools and Fabric

The full list of supported commands can be found in the deploy-tools documentation.

Most fabric commands take the form of

```
fab <environment> <branch> <action>
fab <action that defines its own environment>:arguments
```

Every command in the list of commands is prefixed with `fab`.

If you recieve an error when running your command, make sure that you have run `workon largo-umbrella`, or the name of the Python virtualenv you are using. When run, `workon` will prefix your prompt:

```
you@computer:~$ workon largo
(largo-umbrella)you@computer:~$
```

To exit the virtualenv, you can use the command `deactivate`.

Setting up Largo to contribute documentation

If you just want to help us write documentation, you don't have to go through the complete setup process.

Once you've completed this recipe, you'll be able to:

- rebuild the documentation
- preview your edits in a browser
- rebuild the translation files.
- push your edits to GitHub and request that we incorporate them in Largo

Setting up

This presumes that you're familiar with the command line, and are using OSX or another UNIX-like system. If you're not familiar with the command line, check out our collection of command-line resources.

1. Fork INN/Largo into your own GitHub account.
2. Clone your branch:

   ```
   git clone git@github.com:you/Largo.git
   ```

3. Check out the *write-the-docs* branch:

   ```
   git checkout write-the-docs
   ```

4. Install the required dependencies

   We use some Python libraries to generate our documentation. To install the requirements:

   ```
   cd docs
   ```

   Not required, but it's recommended to install and use virtualenv:

   ```
   mkvirtualenv largo-docs
   workon largo-docs
   ```

   Then:

   ```
   pip install -r requirements.txt
   ```

5. Our API docs/function reference uses doxphp to generate documentation based on the comments embedded in Largo's source code. You'll need to install doxphp to generate API docs.

   - Install with PEAR:

     ```
     pear channel-discover pear.avalanche123.com
     pear install avalanche123/doxphp-beta
     ```

   - Install with git. This requires you to know where your `bin` directory is, and may require `sudo`.

     ```
     git clone https://github.com/avalanche123/doxphp.git
     cd doxphp/bin
     mv doxph* /path/to/bin/
     ```

6. With all dependencies installed, you can run the generator:

   ```
   cd docs
   make php && make html
   ```

   But if you don't want to have to manually recreate the documentation every time you save a file, you can run `grunt watch` from the Largo directory. This command only rebuilds documentation, though, and doesn't recompile the API docs. (For a full list of `grunt` commands, see the Largo grunt docs.

7. You can view the generated docs in the *docs/_build/html* directory:

   There are two main ways of doing this. First, you can view the files with a browser as files. It won't be the best experience.

   The other, better option is to run a sinple web server in the directory that the HTML documentation was output to, and then view them normally as a website in your browser:

```
cd docs/_build/html
python -m SimpleHTTPServer 8081
```

The (ideal) procedure for contributing documentation

1. Choose an issue
2. Comment on the issue that you're taking it.
3. Create a new branch with your contributions, named after the issue:

   git checkout -b 613-partials-sticky-posts
4. Make your changes
5. Commit and push:

   git commit git push -u origin 613-partials-sticky-posts
6. **Create a pull request from your branch to INN/Largo**
   - How to make a PR on GitHub
   - If it's a big PR, please make sure it's well-documented. Thanks!

Resources for writing documentation

- Sphinx' PHP domain-specific markup
- Sphinx reStructuredText primer and quickstart guide

An introduction to the command line

Coursework introductions to the command line:

- CodeCademy: https://www.codecademy.com/learn/learn-the-command-line
- Learn Code The Hard Way: http://cli.learncodethehardway.org/book/
- Thomas Wilburn's CLI Basics: https://github.com/thomaswilburn/itc240-2014/blob/master/assignment-0/command-line.md

Articles

- http://www.davidbaumgold.com/tutorials/command-line/

Cheatsheets

- Matt Waite: https://github.com/mattwaite/JOUR491-News-Applications/blob/master/Helpers/terminal_cheat_sheet.md
- Journalist's Guide to the command line: https://github.com/chrislkeller/nicar15-command-line-basics (Contains many links to more tools and tutorials)
- GitHub 201: https://github.com/githubteacher/nicar-2015
- GitHub Training's cheatsheet: https://training.github.com/kit/downloads/github-git-cheat-sheet

Child Themes

What *is* a child theme?

From the WordPress Codex:

> A child theme is a theme that inherits the functionality and styling of another theme, called the parent theme. Child themes are the recommended way of modifying an existing theme.

Why should you use a child theme?

In order to make it easier to upgrade to future versions of the Largo parent theme, you will want to add any customizations that are unique to your site by creating a child theme. WordPress has a tutorial you can follow that explains how to create and configure a child theme.

More: Using Child Themes.

Using Child Themes

Child themes are a feature of WordPress that allow you to extend and override the parent theme that the child theme is based upon. We encourage you to create a child theme for your project.

Creating Child Themes In order to make it easier to upgrade to future versions of the Largo parent theme, you will want to add any customizations that are unique to your site by creating a child theme. WordPress has a tutorial you can follow that explains how to create and configure a child theme.

NEVER modify the Largo parent theme directly, even to make small changes. It will make your life much harder when we release a new version because your changes are highly likely to be overwritten.

To create a child theme, at a minimum you will need to create a new theme folder (call it something like "largo-child" in your wp-content/themes directory and add one file to it called style.css. That file would be used to add any custom CSS unique to your child theme, but it also tells WordPress where to find the parent theme.

At the very top of the file you need to add at least the following:

```
/*
Theme Name:     Your Child Theme's Name
Theme URI:      Your Site URL
Description:    Your Theme Description
Author:         Your Name
Author URI:     Your Author URL
Template:       largo
Version:        0.1.0
\*/
```

The line starting with "Template" must include the name of the folder that contains the parent theme files (this should be "largo" unless you name the parent theme folder something different).

If you would prefer, we have created an example child theme that you can start from where we document our preferred structure and how to modify the behavior, look and feel of Largo, create custom templates, etc.

To use this child theme, simply download and unzip it just as you did the Largo parent theme above, modify the header block in the style.css as described above and then upload the folder to your wp-content/themes directory along with the Largo parent theme. The sample child theme contains a number of additional files and documentation that you may not need so you might consider removing the elements you do not intend to use.

Now that you have a child theme created, login to your WordPress site and go to Appearance > Themes. Find your child theme, click "activate" and then you should see your new child theme in action on your site and can begin customizing.

Changing Basic Styling Largo has a built-in way to change some basic styling options.

To enable this option, from the *Appearance > Theme Options* screen click on the "Advanced" tab and check the box labelled "Enable Custom LESS to CSS For Theme Customization" and then save the settings.

You will now see an additional menu item under the Appearance menu labelled "CSS Variables". From this menu you will be able to change some basic styling attributes of your Largo site, including the color scheme, fonts and font-sizes.

This option is only intended for making some basic changes to your site's styles. For anything more complex you will need to create a child theme.

Advanced Theme Development and Modification We've created a Largo-Sample-Child-Theme repository to illustrate how we organize child themes that extend Largo.

Visit the repository page to learn more about the following as they pertain to Largo-based child themes:

- Unit Tests and Continuous Integration
- Stylesheets (LESS and CSS)
- Theme Directory Layout
- Custom Theme Javascript or CSS
- Removing or replacing Largo Javascript or CSS

Custom Post Templates

Largo allows you to select between three default templates to use for posts and pages on your site. This default is set from the **Appearance > Theme Options > Layout** tab under the "Single Article Template" heading.

- One Column (Standard Layout) is a new default article template in Largo version 0.4 that focuses on readability, reduces distractions and allows for beautiful presentation of visual elements within a story with a large "hero" section at the top of the article for featured media (photo, video, slideshow or embedded media).
- Two Columns (Classic Layout) is the previous article template from Largo version 0.3 and before which features a content area on the left and a sidebar on the right.
- Full-width (no sidebar) is an option when posts need a wider content area for things like maps and interactive data applications. The full-width template is not ideal for text, as the line length is non-optimal for a good reading experience.

Technical Notes

A few brief technical notes that might be helpful as you get started:

About the `inc/` directory

When you download the theme you'll notice that the /inc folder contains most of the add-on functionality for the parent theme and all of these files are then loaded up via functions.php

Pluggable (overridable) Largo functions

Many of Largo's core functions are pluggable so you can write your own version of them by using the same function name in a child theme's `functions.php`.

You can read up on how that works in the WordPress codex section about child themes.

See: Function Reference.

Theme Options and the Options Framework

Largo uses the Options Framework for the Appearance > Theme Options menu pages.

If you need to access a Theme Options value, use `of_get_option()` instead of the usual `get_option()`. The theme options pages themselves can be customized from `options.php` in the main theme folder.

Homepage Templates

See Homepage layouts from our Largo-Sample-Child-Theme repository.

LESS and CSS

The Largo parent theme uses LESS CSS to generate the stylesheets including a number of elements borrowed and tweaked from Twitter Bootstrap.

You will notice that the theme's main style.css is empty except for the header block because we enqueue our styles from `css/style.css` (the output of /less/style.less when it's compiled), overriding the WordPress default behavior of including the `style.css` file in the root of the theme directory.

TGM Plugin Activation

We use TGM Plugin Activation to package a couple of plugins with the Largo theme that are not currently available in the WordPress plugin directory and to recommend plugins for a number of tasks that are commonly requested for news websites.

- The rest of the theme files and the folder structure should be familiar to most WordPress developers, but if you have any questions, feel free to send us an email at largo@inn.org

Compiling translation files

To rebuild the translation files, run the following commands:

```
grunt pot
msgmerge -o lang/es_ES.po.merged lang/es_ES.po lang/largo.pot
mv lang/es_ES.po.merged lang/es_ES.po
grunt po2mo
```

Images

See the Largo User Guide for Administrators Theme Images page to see the list of image types and sizes you'll need to get your site up and running.

More on image sizes:

- Largo Image Sizes and Constants

Largo Image Sizes and Constants

Largo defines several image sizes for use in various parts of the theme. All sizes are based on a set of constants defined in functions.php. Child themes can override these presets to change the size of images throughout the theme.

Image Sizes The following image sizes are registered in Largo, using the constants defined below.

- **`60x60`**
  - 60x60px image crop.
- **`rect_thumb`**
  - 800x600px image crop
  - Used for cat/tax archive pages.
- **`medium`**
  - Image size defined by constants: `MEDIUM_WIDTHxMEDIUM_HEIGHT`
  - Default is 336px wide by a flexible height.
- **`large`**
  - Image size defined by constants: `LARGE_WIDTHxLARGE_HEIGHT`
  - Default is 771px wide by a flexible height.
- **`full`**
  - Image size defined by constants: `FULL_WIDTHxFULL_HEIGHT`
  - Default is 1170px wide by a flexible height.
- **`third-full`**
  - Image size defined by: `(FULL_WIDTH/3)xFULL_HEIGHT`
  - Default is 390px wide by a flexible height.
- **`two-third-full`**
  - Image size defined by: `(FULL_WIDTH*2/3)xFULL_HEIGHT`
  - Default is 780px wide by a flexible height.

Constants Width:

- `FULL_WIDTH` (default: 1170): the largest width for the largest image size
- `LARGE_WIDTH` (default: 771): medium image crop width
- `MEDIUM_WIDTH` (default: 336): small image crop width

Height:

Largo does not impose any height limits on crop sizes. Thus the defaults are set to 9999.

- `FULL_HEIGHT` (default: 9999): full image crop height
- `LARGE_HEIGHT` (default: 9999): medium image crop height
- `MEDIUM_HEIGHT` (default: 9999): small image crop height

See also Largo's list of constants.

Grunt commands

These commands are run on the command line, prefixed with `grunt`. For example, to rebuild the CSS after editing the LESS files, you would run `grunt less`.

Some commands require you to have external applications installed. Instructions for installing dependencies are included in the developer documentation.

less Recompiles all LESS files into their corresponding CSS files, including sourcemaps, and then runs `cssmin`. The list of files that will be compiled is defined in `Gruntfile.js` in the variable `cssLessFiles`.

uglify Minifies Largo's `.js` JavaScript files to `.min.js` files.

cssmin Takes all `.css` files in `css/` and `homepages/assets/css` that are not `.min.css` files and makes minified versions with the file extension `.min.css`.

shell Runs commands directly on the command line, instead of running Grunt modules.

These commands require you to have set up Largo according to the complete dev environment or documentation contribution environment instructions, because they require several Python libraries that were installed by following those instructions. Besure to have activated your python virtualenv with `workon`, as described in those instructions.

shell:apidocs Recompiles the Largo API Docs from structured comments in Largo's PHP code using /docs/generate_api_docs.py into reStructuredText files.

shell:sphinx Converts all available reStructuredText files into HTML documentation, which is saved locally in `docs/_build/html/`. If you want to preview these docs without pushing them to largo.readthedocs.io, run `python -m SimpleHTTPServer` as described in the documentation contribution instructions.

shell:msmerge Runs msgmerge to merge translation files.

watch Runs `less` if a `.less` file in `less/` or `homepages/assets/less/` is modified. Runs `docs` if a reStructuredText `.rst` file changes in `docs/`.

pot Scans the Largo code for the WordPress localization functions and generates `.po` files for working with localization software.

Running this command requires your computer to have `xgettext` installed. Installation instructions vary based on operating system; your best bet is Google. `xgettext` is usually installed in the same package as `gettext`.

po2mo Converts the `.po` files to `.mo` files. For more information about `.po` and `.mo` files, see the Wikipedia articles on gettext.

Running this command requires your computer to have `xgettext` installed. Installation instructions vary based on operating system; your best bet is Google. `xgettext` is usually installed in the same package as `gettext`.

apidocs Runs `shell:apidocs`, rebuilding only the API docs.

docs Runs `shell:sphinx`, rebuilding ALL docs.

build Runs `less`, `cssmin`, `uglify`, `apidocs`, `docs`, `pot`, and `shell:msmerge` to rebuild the assets, docs, and language files.

version Increments the Largo version number based on `package.json`. Files affected are: `docs/conf.py`, `style.css`, `readme.md`.

build-release Runs `build` and `version`.

publish Runs the following tasks to publish the newest version of Largo on the `master` branch:

confirm Makes sure that you want to publish a release.

gitcheckout Checks out the `master` branch of Largo.

gitmerge Merges the `develop` branch into `master`.

gittag Tags the latest commit with the version number from `package.json`.

gitpush Pushes the `master` branch back to GitHub.

Hooks and filters

Homepage template filters

filter: largo_homepage_feature_stories_list_maximum

Filter the number of posts to display in the list of feature stories in 'HomepageSingleWithFeatured' templates.

args: $max

filter: **largo_homepage_series_stories_list_minimum**

args: $min

Filter the minimum number of posts to show in a series list in the HomepageSingleWithSeriesStories homepage list.

This is used in the query for the series list of posts in the same series as the main feature. If fewer than this number of posts exist, the list is hidden and the headline dominates the full box.

filter: **largo_homepage_series_stories_list_maximum**

args: $max

Filter the maximum number of posts to show in a series list in the HomepageSingleWithSeriesStories homepage list.

This is used in the query for the series list of posts in the same series as the main feature. This is the maximum number of posts that will display in the list.

filter: **largo_homepage_topstories_post_count**

args: $posts_per_page

Filter the number of posts that are displayed in the right-hand side of the Top Stories homepage template.

This is used in the query for the list of posts in the "Homepage Featured" taxonomy. If more than 3 posts are found, they will display under a "More Headlines" heading, just as headline links.

Other filters and actions

filter: **largo_additional_networks**

Called in *inc/widgets/largo-follow.php* and 'inc/header-footer.php' to allow child themes to add additional social networks for social buttons, etc.'_.

post type.

Usage:

function gijn_additional_networks($networks) {

if (of_get_option('listserv_link')) {

$gijn_networks = array('listserv' => 'Join The GIJN Listserv'

); $networks = array_merge($networks, $gijn_networks);

} return $networks;

} add_filter('largo_additional_networks', 'gijn_additional_networks');

filter: **largo_archive_{$post_type}_title**

Called in *archive.php* to filter the page title for posts in the `$post_type` post type.

Usage:

function filter_rounduplink_title($title) { return "Custom title here";

} add_action('largo_archive_rounduplink_title', 'filter_rounduplink_title');

filter: **largo_archive_{$post_type}_feed**

Called in *archive.php* to filter the feed url for posts in the `$post_type` post type.

post type.

Usage:

function filter_column_feed($title) { return "http://example.com/custom_feed_url/feed.xml";

} add_action('largo_archive_column_feed', 'filter_column_feed');

filter: **largo_registration_extra_fields**

Called directly before the *[largo_registration_form]* shortcode has finished executing. You can append to this any addition form fields that you want to process.

Usage:

Passed in is an array of values of post variables generated if a user is trying the form for a second time. You can use these to pre fill your extra field inputs.

Also passed in is a WP_Error object that stores all the generated errors for the page. Use this if you'd like to display an error message on the erroneous field.

```
function filter_function_name ($values, $errors ) {
    # ...
}
add_filter( 'largo_registration_extra_fields', 'filter_function_name' );
```

action: **largo_validate_user_signup_extra_fields**

Called directly before form values from the *[largo_registration_form]*. Hook to this in order to validate any of the extra form data added with the largo_registration_extra_fields filter. For example, you could validate a captcha that was added to the form's fields.

Usage:

Passed in is an array $result which contains all post data for the form. Contained in this array at $result["errors"] is a WP_Error object. Adding errors to this object will cancel form submission.

Also passed in is an array that contains only the extra fields that were present. This is an easy way to check only the extra data.

```
function action_function_name( $result, $extras ) {
    # ...
}
add_action( 'largo_validate_user_signup_extra_fields', 'action_function_name' );
```

filter: **largo_lmp_args**

args: $args

Passed in this are the arguments for the Load More Posts WP_Query. An example usage would be to check if `is_home()` and then restrict the posts returned by the query to those in the homepage featured prominence term.

filter: **largo_lmp_template_partial**

args: $partial, $post_query

Modifies the partial returned by `largo_load_more_posts_choose_partial($post_query)` to whatever you want.

If you are building a custom post type that uses a custom partial, you will need to use this filter to make the correct partial appear in the posts returned by the Load More Posts button on the homepage, on archive pages, and in the search results.

When building your own filter, you must set the fourth parameter of add_filter to 2:

```
function your_filter_name( $partial, $post_type ) {
    // things
    return $partials;
}
add_filter( 'largo_lmp_template_partial', 'your_filter_name', 10, 2 );
                                                               ^
```

Without setting '2', your filter will not be passed the $post_type or $context arguments. In order to set '2', you must set the third parameter of add_filter, which defaults to 10. It is safe to leave that at 10.

filter: **largo_partial_by_post_type**

args: String $partial, String $post_type, String $context

Modifies the partial returned by `largo_get_partial_by_post_type` to whatever you want.

If you are building a custom post type that uses a custom partial, you will need to use this filter to make the correct partial appear in the posts returned by the Load More Posts button on the homepage, on archive pages, and in the search results.

When building your own filter, you must set the fourth parameter of add_filter to 3:

```
function your_filter_name( $partial, $post_type, $context ) {
    // things
    return $partial;
}
add_filter( 'largo_partial_by_post_type', 'your_filter_name', 10, 3 );
                                                               ^
```

Without setting '3', your filter will not be passed the $post_type or $context arguments. In order to set '3', you must set the third parameter of add_filter, which defaults to 10. It is safe to leave that at 10.

filter: largo_byline *args: String $output*

Called in `largo_byline()` before the admin-user edit link is added. This can be used to append or prepend HTML, or to change the output of the byline function entirely. The passed string is HTML.

filter: **largo_post_social_links**

args: String $output

Called before `largo_post_social_links()` returns or echos the social icons. The argument `$output` is HTML, usually containing HTML looking something like this: (Whitespace has been added for readability)

```
<div class="largo-follow post-social clearfix">
    <span class="facebook">
        <a target="_blank" href="http://www.facebook.com/sharer/sharer.php?u=  ...">
            <i class="icon-facebook"></i>
            <span class="hidden-phone">Like</span>
        </a>
    </span>
    <span class="twitter">
        <a target="_blank" href="https://twitter.com/intent/tweet?text= ...">
            <i class="icon-twitter"></i>
            <span class="hidden-phone">Tweet</span>
        </a>
    </span>
    <span class="print">
        <a href="#" onclick="window.print()" title="Print this article" rel="nofollow">
            <i class="icon-print"></i>
            <span class="hidden-phone">Print</span>
        </a>
    </span>
  <span data-service="email" class="email custom-share-button share-button">
        <i class="icon-mail"></i>
        <span class="hidden-phone">Email</span>
    </span>
    <span class="more-social-links">
        <a class="popover-toggle" href="#"><i class="icon-plus"></i><span class="hidden-phone">M
        <span class="popover">
        <ul>
            ${more_social_links_str}
        </ul>
        </span>
    </span>
</div>
```

filter: largo_post_social_more_social_links *args: Array $more_social_links*

Called in *largo_post_social_links* to filter the array of social links in the "More" drop-down menu displayed in the social links on single posts: the article-top social links, the floating social buttons, and the Largo Follow widget in the article-bottom widget area.

Passed is an array, where each item in the array is an HTML *li* element containing a link and an icon *i* element to some form of additional, relevant material. The default array in Largo is:

- Top term link
- Subscribe to RSS feed for top term
- Author Twitter link, if the post doesn't have a custom byline and if Co-Authors Plus isn't active

Adding new social media networks is as simple as adding a new item to the array:

```
function add_linkedin( $more ) {
    $more[] = '<li><a href=""><i class="icon-linkedin"></i> <span>Your text here!</span></a></li
    return $more;
}
add_filter( 'largo_post_social_more_social_links', 'add_linkedin' );
```

filter largo_remove_hero

Filter to disable largo_remove_hero based on the global $post at the time the function is run

Since 0.5.5

Parameters

- **$run** (*Boolean*) – Whether the function should run against the current post
- **$post** (*WP_Post*) – The global `$post` object at the time the function is run

When building your own filter, you must set the fourth parameter of add_filter to 2:

```
function filter_largo_remove_hero( $run, $post ) {
    # determine whether or not to run largo_remove_hero based on $post
    return $run;
}
add_filter( 'largo_remove_hero', 'filter_largo_remove_hero', 10, 2 );
                                                              ^
```

filter largo_top_term_metabox_taxonomies

Called in the `largo_top_tag_display` metabox to allow themes to filter the taxonomies from which are drawn the term options for the top term metabox display.

Since 0.5.5

Parameters

- **$taxonomies** (*Array*) – array('series', 'category', 'post_tag', 'prominence')

Add new taxonomies like so:

```
function add_taxonomies( $taxonomies ) {
    $taxonomies[] = 'columns';
    $taxonomies[] = 'post-type';
    return $taxonomies;
}
add_filter('largo_top_term_metabox_taxonomies', 'add_taxonomies');
```

Template Hooks

What are these and why would I want to use them?

Sometimes you may want to fire certain functions or include additional blocks of markup on a page without having to modify or override an entire template file.

WordPress allows you to define custom action hooks using the do_action() function like so:

```
do_action( 'largo_top' );
```

and then from functions.php in a child theme you can use the add_action() function to fire another function you define to insert markup or perform some other action when the do_action() function is executed, for example:

```
add_action( 'largo_top', 'largo_render_network_header' );
```

This usage would call the `largo_render_network_header` function when the largo_top action is executed.

We are in the process of adding a number of action hooks to Largo to make it easier for developers to modify templates without having to completely replace them in a child theme.

This has the advantage of making your code much easier to maintain (because you're more explicit about what part of the template you're modifying) and also makes it easier to make the update to future versions of Largo because even if the template files change considerably, the placement of the hooks will likely remain the same.

Here is the current list of hooks available in Largo (available as of v.0.4):

header.php

- **(wp_head)** - if you need to insert anything in the <head> element use the built-in wp_head hook
- **largo_top** - directly after the opening <body> tag and "return to top" target div
- **largo_before_global_nav** - only fires if the global nav is shown, directly before the global nav partial
- **largo_after_global_nav** - only fires if the global nav is shown, directly after the global nav partial
- **largo_before_header** - before the main <header> element
- **largo_after_header** - after the main <header> element
- **largo_after_nav** - after the nav, before #main opening div tag
- **largo_main_top** - directly after the opening #main div tag

partials/largo-header.php

- **largo_header_before_largo_header** - immediately before `largo_header()` is output
- **largo_header_after_largo_header** - immediately after `largo_header()` is output. By default, `largo_header_widget_sidebar` is hooked here.

for all lists of posts

- **largo_loop_after_post_x** - fires after every post in a river of posts on the homepage or archive pages. This is helpful if you want to insert interstitial content in a river of posts (typically things like newsletter subscription widgets, donation messages, etc.).

This action takes a couple of arguments that may come in handy:

do_action('largo_loop_after_post_x', $counter, $context);

- **$counter** tracks the number of posts in any given loop
- **$context** is presently either 'archive' or 'home' to give you flexibility to insert different interstitials for different page types.

an example of this in use might look like:

function mytheme_interstitial($counter, $context) {

if ($counter === 2 && $context === 'home') { // do homepage stuff

} elseif ($counter === 2 && $context === 'archive') { // do something different in the same spot on archive pages

}

} add_action('largo_loop_after_post_x', 'mytheme_interstitial', 10, 2);

home.php

These actions are run on all homepage templates, including the Legacy Three Column layout.

- **largo_before_sticky_posts** - Runs in the main column, before the sticky post would be rendered
- **largo_after_sticky_posts** - Runs in the main column, after where the sticky post would be rendered, before the homepage bottom area.
- **largo_after_homepage_hottom** - Runs after the homepage bottom area, before the termination of the main column.

sidebar.php

- **largo_before_sidebar** - before the sidebar opening div tag
- **largo_before_sidebar_widgets** - after the opening div tag but before the first widget
- **largo_after_sidebar_widgets** - after the last widget but before the closing div tag
- **largo_after_sidebar** - after the closing div tag

footer.php

- **largo_before_footer** - after the closing div tag for #page but before the .footer-bg (this also comes after the optional "before footer" widget area that can be activated from the layout tab of the theme options
- **largo_before_footer_widgets** - before the main footer widget areas
- **largo_before_footer_boilerplate** - after the main footer widget areas and before the boilerplate (copyright message, credits, etc.)
- **largo_after_footer_copyright** - after the copyright message paragraph, but before the end of the boilerplate; useful if you want to insert addresses or other information about your site
- **largo_before_footer_close** - after the boilerplate but still inside the footer container
- **largo_after_footer** - after the closing tag for .footer-bg but before the sticky footer
- **(wp_footer)** - if you need to insert anything just before the closing <body> tag use the wp_footer hook

single.php

- **largo_before_post_header** - inside <article> but before the post <header> element
- **largo_after_post_header** - just after the closing post <header> element (before the hero image/video)
- **largo_after_hero - in the single column** (new) single post template, just after the hero (featured) image/video
- **largo_after_post_content** - directly after the .entry-content closing tag
- **largo_after_post_footer** (deprecated in 0.4) - before the closing </article> tag, replaced in the new layouts by largo_after_post_content
- **largo_before_post_bottom_widget_area** - after the closing </article> tag but before the post bottom widget area
- **largo_post_bottom_widget_area** - by default, the "Article Bottom" widget area is output here through *largo_post_bottom_widget_area*
- **largo_after_post_bottom_widget_area** - directly after the post bottom widget area (but before the comments section)
- **largo_before_comments** - before the comments section
- **largo_after_comments** - after the comments section
- **largo_after_content** - after the close of the #content div

page.php

- **largo_before_page_header** - inside <article> but before the post <header> element
- **largo_after_page_header** - just after the closing post <header> element
- **largo_before_page_content** - directly inside the .entry-content tag
- **largo_after_page_content** - directly before the .entry-content closing tag

category.php

- **largo_category_after_description_in_header** - between the `div.archive-description` and before `get_template_part('partials/archive', 'category-related');`.

- **largo_before_category_river** - just before the river of stories at the bottom of the category archive page (for adding a header to this column, for example)
- **largo_loop_after_post_x** - runs after every post, with arguments `$counter` and `context` describing which post it's running after and what the context is. (In categories, the context is `archive`.)
- **largo_after_category_river** - immediately after the river of stories at the bottom of the category archive page, after the Load More Posts button (for adding a footer to this column, for example.)

search.php

The Largo search page has two main modes: Google Custom Search Engine and the standard WordPress search emgine. Because the dispalyed layouts are so different, each has their own set of actions.

- **largo_search_gcs_before_container**: If Google Custom Search is enabled, fires before the GCS container
- **largo_search_gcs_after_container**: If Google Custom Search is enabled, fires after the GCS container
- **largo_search_normal_before_form**: Fires before the ouput from `get_search_form()`
- **largo_search_normal_before_results**: Fires between `get_search_from` and "Your search for %s returned %s results", and runs even if there were no search results.
- **largo_search_normal_after_results**: Fires after the search results or `partials/content-not-found` are displayed.

Function Reference

NOTE: the function reference is a work-in-progress and may not be very useful at the moment.

It may be helpful to read Largo's source on Github. If you discover insight there that is not included in these docs, please let us know.

Function reference by file

- archive.php
- feed-mailchimp.php
- functions.php
- options.php
- homepages/homepage.php
- homepages/layouts/HomepageSingleWithSeriesStories.php
- homepages/zones/zones.php
- inc/ajax-functions.php
- inc/avatars.php
- inc/byline_class.php
- inc/conditionals.php
- inc/custom-feeds.php
- inc/custom-less-variables.php
- inc/deprecated.php
- inc/editor.php

- inc/enqueue.php
- inc/featured-content.php
- inc/featured-media.php
- inc/header-footer.php
- inc/helpers.php
- inc/images.php
- inc/metabox-api.php
- inc/nav-menus.php
- inc/post-metaboxes.php
- inc/post-social.php
- inc/post-tags.php
- inc/post-templates.php
- inc/related-content.php
- inc/sidebars.php
- inc/taxonomies.php
- inc/term-icons.php
- inc/term-meta.php
- inc/term-sidebars.php
- inc/update.php
- inc/users.php
- inc/verify.php
- inc/widgets.php
- inc/avatars/functions.php
- inc/customizer/customizer.php
- inc/widgets/largo-author-bio.php
- inc/widgets/largo-facebook.php
- inc/widgets/largo-image-widget.php
- inc/widgets/largo-post-series-links.php
- inc/widgets/largo-recent-comments.php
- inc/widgets/largo-recent-posts.php
- inc/widgets/largo-taxonomy-list.php
- inc/widgets/largo-twitter.php
- inc/wp-taxonomy-landing/functions/cftl-admin.php
- inc/wp-taxonomy-landing/functions/cftl-series-order.php

Largo Constants

The image constants

Largo's image constants are used to define the crop and scaling sizes that WordPress automatically chops your image into.

Width:

- `FULL_WIDTH` (default: 1170): the largest width for the largest image size
- `LARGE_WIDTH` (default: 771): medium image crop width
- `MEDIUM_WIDTH` (default: 336): small image crop width

Height:

Largo does not impose any height limits on crop sizes. Thus the defaults are set to 9999.

- `FULL_HEIGHT` (default: 9999): full image crop height
- `LARGE_HEIGHT` (default: 9999): medium image crop height
- `MEDIUM_HEIGHT` (default: 9999): small image crop height

For more information about how Largo handles image sizes, see Image Sizes.

The other constants

constant LARGO_DEBUG

`LARGO_DEBUG` should be set to `true` in development environments. It controls many behaviors:

- **in `inc/enqueue.php`, `LARGO_DEBUG` controls whether or not minified versions of the following files are used:**
  - `css/style.css`
  - `js/largoCore.js`
  - `css/widgets-php.css`
  - `js/widgets-php.js`
- in `inc/custom-less-variables.php`, `LARGO_DEBUG` controls whether or not minified versions of the recompiled files are used.
- **in `inc/featured-media.php`, `LARGO_DEBUG` controls whether or not minified versions of the following files are**
  - `js/featured-media.js`
- **in `inc/post-metaboxes.php`, `LARGO_DEBUG` controls whether or not minified versions of the following files are**
  - `js/custom-sidebar.js`
  - `js/top-terms.js`
- **in `inc/term-icons.php`, `LARGO_DEBUG` controls whether or not minified versions of the following files are used:**
  - `js/custom-term-icons.js`
- **in `inc/update.php`,**

```
        - js/update-page.js
```

Define `LARGO_DEBUG` to `true` in your `wp-config.php` with the following line:

```
define( 'LARGO_DEBUG', TRUE);
```

Please be careful with `LARGO_DEBUG`-related functionality, as it is difficult to write tests for functions including constants.

constant OPTIONS_FRAMEWORK_DIRECTORY
This constant represents the URI of the options framework, defined as `get_template_directory_uri() . '/lib/options-framework/'` in `functions.php`. This path is used to enqueue the options framework CSS, color picker CSS, jquery-dependent color picker, iris.min.js, the options framework scripts, and the options framework media library uploader.

constant SHOW_GLOBAL_NAV
The Global Nav is a thin blck bar displayed in the header of Largo, goverened by `SHOW_GLOBAL_NAV`. `SHOW_GLOBAL_NAV` defaults to true, but child themes can set it to false with `define( 'SHOW_GLOBAL_NAV', FALSE );` in their theme `functions.php`.

constant SHOW_STICKY_NAV
DEPRECATED in Largo version 0.5.5. Conditional logic based on this constant should remove the conditional logic, and make sure that the HTML stucture is similar to that of partials/nav_sticky.php. The element `#sticky-nav-holder` will be shown or hidden by navigation.js.

The sticky nav used to appear on the homepage and all internal pages, and on mobile devices, governed by `SHOW_STICKY_NAV`. `SHOW_STICKY_NAV` may be defined to be true or false.

constant SHOW_MAIN_NAV
The main navigation appears on the homepage and all internal pages, but not on mobile devices, governed by `SHOW_MAIN_NAV`. `SHOW_MAIN_NAV` defaults to true, but child themes can set it to false with `define( 'SHOW_GLOBAL_NAV', FALSE );` in their theme ‘‘functions.

constant SHOW_SECONDARY_NAV

constant SHOW_CATEGORY_RELATED_TOPICS

constant LARGO_AVATAR_META_NAME

constant LARGO_AVATAR_ACTION_NAME

constant LARGO_AVATAR_INPUT_NAME

constant JCLV_UNCOMPRESSED

constant DOING_AUTOSAVE

constant PICTUREFILL_WP_PATH

constant PICTUREFILL_WP_URL

constant PICTUREFILL_WP_VERSION

constant CFTL_SELF_DIR

constant LARGO_TEMPLATE_LANDING_VERSION

constant MEDIA_CREDIT_POSTMETA_KEY

Hooks and filters

Homepage template filters

filter: largo_homepage_feature_stories_list_maximum

Filter the number of posts to display in the list of feature stories in 'HomepageSingleWithFeatured' templates.

args: $max

filter: **largo_homepage_series_stories_list_minimum**

args: $min

Filter the minimum number of posts to show in a series list in the HomepageSingleWithSeriesStories homepage list.

This is used in the query for the series list of posts in the same series as the main feature. If fewer than this number of posts exist, the list is hidden and the headline dominates the full box.

filter: **largo_homepage_series_stories_list_maximum**

args: $max

Filter the maximum number of posts to show in a series list in the HomepageSingleWithSeriesStories homepage list.

This is used in the query for the series list of posts in the same series as the main feature. This is the maximum number of posts that will display in the list.

filter: **largo_homepage_topstories_post_count**

args: $posts_per_page

Filter the number of posts that are displayed in the right-hand side of the Top Stories homepage template.

This is used in the query for the list of posts in the "Homepage Featured" taxonomy. If more than 3 posts are found, they will display under a "More Headlines" heading, just as headline links.

Other filters and actions

filter: **largo_additional_networks**

Called in *inc/widgets/largo-follow.php* and 'inc/header-footer.php' to allow child themes to add additional social networks for social buttons, etc.'_.

post type.

Usage:

function gijn_additional_networks($networks) {

if (of_get_option('listserv_link')) {

$gijn_networks = array('listserv' => 'Join The GIJN Listserv'

); $networks = array_merge($networks, $gijn_networks);

} return $networks;

} add_filter('largo_additional_networks', 'gijn_additional_networks');

filter: **largo_archive_{$post_type}_title**

Called in *archive.php* to filter the page title for posts in the `$post_type` post type.

Usage:

function filter_rounduplink_title($title) { return "Custom title here";

} add_action('largo_archive_rounduplink_title', 'filter_rounduplink_title');

filter: **largo_archive_{$post_type}_feed**

Called in *archive.php* to filter the feed url for posts in the `$post_type` post type.

post type.

Usage:

function filter_column_feed($title) { return "http://example.com/custom_feed_url/feed.xml";

} add_action('largo_archive_column_feed', 'filter_column_feed');

filter: **largo_registration_extra_fields**

Called directly before the *[largo_registration_form]* shortcode has finished executing. You can append to this any addition form fields that you want to process.

Usage:

Passed in is an array of values of post variables generated if a user is trying the form for a second time. You can use these to pre fill your extra field inputs.

Also passed in is a WP_Error object that stores all the generated errors for the page. Use this if you'd like to display an error message on the erroneous field.

```
function filter_function_name ($values, $errors ) {
    # ...
}
add_filter( 'largo_registration_extra_fields', 'filter_function_name' );
```

action: **largo_validate_user_signup_extra_fields**

Called directly before form values from the *[largo_registration_form]*. Hook to this in order to validate any of the extra form data added with the largo_registration_extra_fields filter. For example, you could validate a captcha that was added to the form's fields.

Usage:

Passed in is an array $result which contains all post data for the form. Contained in this array at $result["errors"] is a WP_Error object. Adding errors to this object will cancel form submission.

Also passed in is an array that contains only the extra fields that were present. This is an easy way to check only the extra data.

```
function action_function_name( $result, $extras ) {
    # ...
}
add_action( 'largo_validate_user_signup_extra_fields', 'action_function_name' );
```

filter: **largo_lmp_args**

args: $args

Passed in this are the arguments for the Load More Posts WP_Query. An example usage would be to check if `is_home()` and then restrict the posts returned by the query to those in the homepage featured prominence term.

filter: **largo_lmp_template_partial**

args: $partial, $post_query

Modifies the partial returned by `largo_load_more_posts_choose_partial($post_query)` to whatever you want.

If you are building a custom post type that uses a custom partial, you will need to use this filter to make the correct partial appear in the posts returned by the Load More Posts button on the homepage, on archive pages, and in the search results.

When building your own filter, you must set the fourth parameter of add_filter to 2:

```
function your_filter_name( $partial, $post_type ) {
    // things
    return $partials;
}
add_filter( 'largo_lmp_template_partial', 'your_filter_name', 10, 2 );
                                                                ^
```

Without setting '2', your filter will not be passed the $post_type or $context arguments. In order to set '2', you must set the third parameter of add_filter, which defaults to 10. It is safe to leave that at 10.

filter: **largo_partial_by_post_type**

args: String $partial, String $post_type, String $context

Modifies the partial returned by `largo_get_partial_by_post_type` to whatever you want.

If you are building a custom post type that uses a custom partial, you will need to use this filter to make the correct partial appear in the posts returned by the Load More Posts button on the homepage, on archive pages, and in the search results.

When building your own filter, you must set the fourth parameter of add_filter to 3:

```
function your_filter_name( $partial, $post_type, $context ) {
    // things
    return $partial;
}
add_filter( 'largo_partial_by_post_type', 'your_filter_name', 10, 3 );
                                                                ^
```

Without setting '3', your filter will not be passed the $post_type or $context arguments. In order to set '3', you must set the third parameter of add_filter, which defaults to 10. It is safe to leave that at 10.

filter: largo_byline *args: String $output*

Called in `largo_byline()` before the admin-user edit link is added. This can be used to append or prepend HTML, or to change the output of the byline function entirely. The passed string is HTML.

filter: **largo_post_social_links**

args: String $output

Called before `largo_post_social_links()` returns or echos the social icons. The argument `$output` is HTML, usually containing HTML looking something like this: (Whitespace has been added for readability)

```
<div class="largo-follow post-social clearfix">
    <span class="facebook">
        <a target="_blank" href="http://www.facebook.com/sharer/sharer.php?u=  ...">
            <i class="icon-facebook"></i>
            <span class="hidden-phone">Like</span>
        </a>
    </span>
    <span class="twitter">
        <a target="_blank" href="https://twitter.com/intent/tweet?text= ...">
```

```
                <i class="icon-twitter"></i>
                <span class="hidden-phone">Tweet</span>
            </a>
        </span>
        <span class="print">
            <a href="#" onclick="window.print()" title="Print this article" rel="nofollow">
                <i class="icon-print"></i>
                <span class="hidden-phone">Print</span>
            </a>
        </span>
      <span data-service="email" class="email custom-share-button share-button">
            <i class="icon-mail"></i>
            <span class="hidden-phone">Email</span>
        </span>
        <span class="more-social-links">
            <a class="popover-toggle" href="#"><i class="icon-plus"></i><span class="hidden-phone">M
            <span class="popover">
            <ul>
                ${more_social_links_str}
            </ul>
            </span>
        </span>
    </div>
```

filter: largo_post_social_more_social_links *args: Array $more_social_links*

Called in *largo_post_social_links* to filter the array of social links in the "More" drop-down menu displayed in the social links on single posts: the article-top social links, the floating social buttons, and the Largo Follow widget in the article-bottom widget area.

Passed is an array, where each item in the array is an HTML *li* element containing a link and an icon *i* element to some form of additional, relevant material. The default array in Largo is:

- Top term link
- Subscribe to RSS feed for top term
- Author Twitter link, if the post doesn't have a custom byline and if Co-Authors Plus isn't active

Adding new social media networks is as simple as adding a new item to the array:

```
function add_linkedin( $more ) {
    $more[] = '<li><a href=""><i class="icon-linkedin"></i> <span>Your text here!</span></a></li
    return $more;
}
add_filter( 'largo_post_social_more_social_links', 'add_linkedin' );
```

filter largo_remove_hero

Filter to disable largo_remove_hero based on the global $post at the time the function is run

Since 0.5.5

Parameters

- **$run** (*Boolean*) – Whether the function should run against the current post
- **$post** (*WP_Post*) – The global `$post` object at the time the function is run

When building your own filter, you must set the fourth parameter of add_filter to 2:

```
function filter_largo_remove_hero( $run, $post ) {
    # determine whether or not to run largo_remove_hero based on $post
```

```
    return $run;
}
add_filter( 'largo_remove_hero', 'filter_largo_remove_hero', 10, 2 );
                                                              ^
```

filter largo_top_term_metabox_taxonomies

Called in the `largo_top_tag_display` metabox to allow themes to filter the taxonomies from which are drawn the term options for the top term metabox display.

Since 0.5.5

Parameters

- **$taxonomies** (*Array*) – array('series', 'category', 'post_tag', 'prominence')

Add new taxonomies like so:

```
function add_taxonomies( $taxonomies ) {
    $taxonomies[] = 'columns';
    $taxonomies[] = 'post-type';
    return $taxonomies;
}
add_filter('largo_top_term_metabox_taxonomies', 'add_taxonomies');
```

Template Hooks

What are these and why would I want to use them?

Sometimes you may want to fire certain functions or include additional blocks of markup on a page without having to modify or override an entire template file.

WordPress allows you to define custom action hooks using the do_action() function like so:

```
do_action( 'largo_top' );
```

and then from functions.php in a child theme you can use the add_action() function to fire another function you define to insert markup or perform some other action when the do_action() function is executed, for example:

```
add_action( 'largo_top', 'largo_render_network_header' );
```

This usage would call the `largo_render_network_header` function when the largo_top action is executed.

We are in the process of adding a number of action hooks to Largo to make it easier for developers to modify templates without having to completely replace them in a child theme.

This has the advantage of making your code much easier to maintain (because you're more explicit about what part of the template you're modifying) and also makes it easier to make the update to future versions of Largo because even if the template files change considerably, the placement of the hooks will likely remain the same.

Here is the current list of hooks available in Largo (available as of v.0.4):

header.php

- **(wp_head)** - if you need to insert anything in the <head> element use the built-in wp_head hook
- **largo_top** - directly after the opening <body> tag and "return to top" target div
- **largo_before_global_nav** - only fires if the global nav is shown, directly before the global nav partial
- **largo_after_global_nav** - only fires if the global nav is shown, directly after the global nav partial
- **largo_before_header** - before the main <header> element

- **largo_after_header** - after the main <header> element
- **largo_after_nav** - after the nav, before #main opening div tag
- **largo_main_top** - directly after the opening #main div tag

partials/largo-header.php

- **largo_header_before_largo_header** - immediately before `largo_header()` is output
- **largo_header_after_largo_header** - immediately after `largo_header()` is output. By default, `largo_header_widget_sidebar` is hooked here.

for all lists of posts

- **largo_loop_after_post_x** - fires after every post in a river of posts on the homepage or archive pages. This is helpful if you want to insert interstitial content in a river of posts (typically things like newsletter subscription widgets, donation messages, etc.).

This action takes a couple of arguments that may come in handy:

do_action('largo_loop_after_post_x', $counter, $context);

- **$counter** tracks the number of posts in any given loop
- **$context** is presently either 'archive' or 'home' to give you flexibility to insert different interstitials for different page types.

an example of this in use might look like:

function mytheme_interstitial($counter, $context) {

if ($counter === 2 && $context === 'home') { // do homepage stuff

} elseif ($counter === 2 && $context === 'archive') { // do something different in the same spot on archive pages

}

} add_action('largo_loop_after_post_x', 'mytheme_interstitial', 10, 2);

home.php

These actions are run on all homepage templates, including the Legacy Three Column layout.

- **largo_before_sticky_posts** - Runs in the main column, before the sticky post would be rendered
- **largo_after_sticky_posts** - Runs in the main column, after where the sticky post would be rendered, before the homepage bottom area.
- **largo_after_homepage_hottom** - Runs after the homepage bottom area, before the termination of the main column.

sidebar.php

- **largo_before_sidebar** - before the sidebar opening div tag
- **largo_before_sidebar_widgets** - after the opening div tag but before the first widget
- **largo_after_sidebar_widgets** - after the last widget but before the closing div tag
- **largo_after_sidebar** - after the closing div tag

footer.php

- **largo_before_footer** - after the closing div tag for #page but before the .footer-bg (this also comes after the optional "before footer" widget area that can be activated from the layout tab of the theme options
- **largo_before_footer_widgets** - before the main footer widget areas

- **largo_before_footer_boilerplate** - after the main footer widget areas and before the boilerplate (copyright message, credits, etc.)
- **largo_after_footer_copyright** - after the copyright message paragraph, but before the end of the boilerplate; useful if you want to insert addresses or other information about your site
- **largo_before_footer_close** - after the boilerplate but still inside the footer container
- **largo_after_footer** - after the closing tag for .footer-bg but before the sticky footer
- **(wp_footer)** - if you need to insert anything just before the closing <body> tag use the wp_footer hook

single.php

- **largo_before_post_header** - inside <article> but before the post <header> element
- **largo_after_post_header** - just after the closing post <header> element (before the hero image/video)
- **largo_after_hero - in the single column** (new) single post template, just after the hero (featured) image/video
- **largo_after_post_content** - directly after the .entry-content closing tag
- **largo_after_post_footer** (deprecated in 0.4) - before the closing </article> tag, replaced in the new layouts by largo_after_post_content
- **largo_before_post_bottom_widget_area** - after the closing </article> tag but before the post bottom widget area
- **largo_post_bottom_widget_area** - by default, the "Article Bottom" widget area is output here through *largo_post_bottom_widget_area*
- **largo_after_post_bottom_widget_area** - directly after the post bottom widget area (but before the comments section)
- **largo_before_comments** - before the comments section
- **largo_after_comments** - after the comments section
- **largo_after_content** - after the close of the #content div

page.php

- **largo_before_page_header** - inside <article> but before the post <header> element
- **largo_after_page_header** - just after the closing post <header> element
- **largo_before_page_content** - directly inside the .entry-content tag
- **largo_after_page_content** - directly before the .entry-content closing tag

category.php

- **largo_category_after_description_in_header** - between the `div.archive-description` and before `get_template_part('partials/archive', 'category-related');`.
- **largo_before_category_river** - just before the river of stories at the bottom of the category archive page (for adding a header to this column, for example)
- **largo_loop_after_post_x** - runs after every post, with arguments `$counter` and `context` describing which post it's running after and what the context is. (In categories, the context is `archive`.)
- **largo_after_category_river** - immediately after the river of stories at the bottom of the category archive page, after the Load More Posts button (for adding a footer to this column, for example.)

search.php

The Largo search page has two main modes: Google Custom Search Engine and the standard WordPress search emgine. Because the dispalyed layouts are so different, each has their own set of actions.

- **largo_search_gcs_before_container**: If Google Custom Search is enabled, fires before the GCS container
- **largo_search_gcs_after_container**: If Google Custom Search is enabled, fires after the GCS container
- **largo_search_normal_before_form**: Fires before the ouput from `get_search_form()`
- **largo_search_normal_before_results**: Fires between `get_search_from` and "Your search for %s returned %s results", and runs even if there were no search results.
- **largo_search_normal_after_results**: Fires after the search results or `partials/content-not-found` are displayed.

Bug Reports and Feature Requests

Our preferred way for you to submit bug reports, requests for new features or even questions about how things work in Largo is by opening a new issue on the Largo github repository.

Contributing to Largo

We welcome (and encourage) anyone who wants to contribute back to the project.

To begin, please review our contribution guidelines.

We have many ways you can contribute and not all are technical. Wherever possible we will flag issues that we believe are good for beginners or for less/non-technical contributors (writing/improving documentation, etc.).

Our roadmap, open issues, suggested features and discussion can always be found in the issues section of the Largo github repository.

We also have documentation on the Anatomy of a Pull Request and Submission Protocol and Contributing to the INN Nerds docs repo using Github.com which explain, at a high level, the process of contributing to Github projects, generally.

If you would like to help with the documentation, here are some resources:

- Sphinx' PHP domain-specific markup
- Sphinx reStructuredText primer and quickstart guide

If you have feedback on this collection of documentation, please get in touch.

Largo Constants

The image constants

Largo's image constants are used to define the crop and scaling sizes that WordPress automatically chops your image into.

Width:

- `FULL_WIDTH` (default: 1170): the largest width for the largest image size
- `LARGE_WIDTH` (default: 771): medium image crop width
- `MEDIUM_WIDTH` (default: 336): small image crop width

Height:

Largo does not impose any height limits on crop sizes. Thus the defaults are set to 9999.

- FULL_HEIGHT (default: 9999): full image crop height
- LARGE_HEIGHT (default: 9999): medium image crop height
- MEDIUM_HEIGHT (default: 9999): small image crop height

For more information about how Largo handles image sizes, see Image Sizes.

The other constants

constant LARGO_DEBUG

LARGO_DEBUG should be set to true in development environments. It controls many behaviors:

- **in inc/enqueue.php, LARGO_DEBUG controls whether or not minified versions of the following files are used:**
  - css/style.css
  - js/largoCore.js
  - css/widgets-php.css
  - js/widgets-php.js
- in inc/custom-less-variables.php, LARGO_DEBUG controls whether or not minified versions of the recompiled files are used.
- **in inc/featured-media.php, LARGO_DEBUG controls whether or not minified versions of the following files are**
  - js/featured-media.js
- **in inc/post-metaboxes.php, LARGO_DEBUG controls whether or not minified versions of the following files are**
  - js/custom-sidebar.js
  - js/top-terms.js
- **in inc/term-icons.php, LARGO_DEBUG controls whether or not minified versions of the following files are used:**
  - js/custom-term-icons.js
- **in inc/update.php,**
  - js/update-page.js

Define LARGO_DEBUG to true in your wp-config.php with the following line:

```
define( 'LARGO_DEBUG', TRUE);
```

Please be careful with LARGO_DEBUG-related functionality, as it is difficult to write tests for functions including constants.

constant OPTIONS_FRAMEWORK_DIRECTORY

This constant represents the URI of the options framework, defined as get_template_directory_uri() . '/lib/options-framework/' in functions.php. This path is used to enqueue the options framework CSS, color picker CSS, jquery-dependent color picker, iris.min.js, the options framework scripts, and the options framework media library uploader.

constant SHOW_GLOBAL_NAV

The Global Nav is a thin blck bar displayed in the header of Largo, goverened by SHOW_GLOBAL_NAV.

SHOW_GLOBAL_NAV defaults to true, but child themes can set it to false with `define( 'SHOW_GLOBAL_NAV', FALSE );` in their theme `functions.php`.

constant SHOW_STICKY_NAV

DEPRECATED in Largo version 0.5.5. Conditional logic based on this constant should remove the conditional logic, and make sure that the HTML stucture is similar to that of partials/nav_sticky.php. The element `#sticky-nav-holder` will be shown or hidden by navigation.js.

The sticky nav used to appear on the homepage and all internal pages, and on mobile devices, governed by SHOW_STICKY_NAV. SHOW_STICKY_NAV may be defined to be true or false.

constant SHOW_MAIN_NAV

The main navigation appears on the homepage and all internal pages, but not on mobile devices, governed by SHOW_MAIN_NAV. SHOW_MAIN_NAV defaults to true, but child themes can set it to false with `define( 'SHOW_GLOBAL_NAV', FALSE );` in their theme ‘‘functions.

constant SHOW_SECONDARY_NAV

constant SHOW_CATEGORY_RELATED_TOPICS

constant LARGO_AVATAR_META_NAME

constant LARGO_AVATAR_ACTION_NAME

constant LARGO_AVATAR_INPUT_NAME

constant JCLV_UNCOMPRESSED

constant DOING_AUTOSAVE

constant PICTUREFILL_WP_PATH

constant PICTUREFILL_WP_URL

constant PICTUREFILL_WP_VERSION

constant CFTL_SELF_DIR

constant LARGO_TEMPLATE_LANDING_VERSION

constant MEDIA_CREDIT_POSTMETA_KEY

Feedback on these docs

If you have comments or suggestions on this documentation, you can reach us through the following ways:

- File an issue on Largo’s Github repository
- Send an email to nerds@inn.org
- Tweet to us @innnerds

C

D

J

L

M

O

P

S