Community Experience Distilled

Elgg 1.8 Social Networking

Create, customize, and deploy your very own social networking site with Elgg

Foreword by Dave Tosh, Elgg co-founder

Cash Costello

[PACKT] open source*
PUBLISHING
community experience distilled

Elgg 1.8 Social Networking

Create, customize, and deploy your very own social networking site with Elgg

Cash Costello

[PACKT] PUBLISHING open source community experience distilled

BIRMINGHAM - MUMBAI

Elgg 1.8 Social Networking

Copyright © 2012 Packt Publishing

All rights reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior written permission of the publisher, except in the case of brief quotations embedded in critical articles or reviews.

Every effort has been made in the preparation of this book to ensure the accuracy of the information presented. However, the information contained in this book is sold without warranty, either express or implied. Neither the author, nor Packt Publishing, and its dealers and distributors will be held liable for any damages caused or alleged to be caused directly or indirectly by this book.

Packt Publishing has endeavored to provide trademark information about all of the companies and products mentioned in this book by the appropriate use of capitals. However, Packt Publishing cannot guarantee the accuracy of this information.

First published: March 2008

Second edition: February 2012

Production Reference: 2130212

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84951-130-8

`www.packtpub.com`

Cover Image by Jose (`jose@joseargudo.com`)

Credits

Author
Cash Costello

Author of 1st Edition
Mayank Sharma

Reviewers
Valentin Crettaz
Kevin Jardine
Danny Lieberman
Marcus Povey
Brett Profitt
Liran Tal
Evan Winslow

Acquisition Editor
David Barnes

Lead Technical Editor
Meeta Rajani

Technical Editor
Llewellyn F. Rozario

Project Coordinator
Vishal Bodwani

Proofreader
Aaron Nash

Indexer
Hemangini Bari

Production Coordinator
Arvindkumar Gupta

Cover Work
Arvindkumar Gupta

Foreword

I am delighted that Cash Costello undertook the task of writing this book. Cash typifies the very essence of open source. He contributes on every level to the Elgg project: bug reports, core patches, plugins as well as offering advice and support in the community. Therefore, it is fitting that it is he who authors this edition.

As an updated version of the first Elgg book, this is an excellent resource for those interested in Elgg development due to its attention to detail, clearly written style and knowledgeable author.

I would like to give a special mention to Brett Profitt, Elgg's lead developer, and the technical reviewer of this book. Brett has played a key role in the continually improvement of Elgg and over the past 20 months or so his efforts have had a hugely positive impact on the wider Elgg community with more members now participating in Elgg's development, promotion, and support.

Elgg has come a long way from the very first version. Having started as a proof-of-concept, Elgg has grown into a leading social networking engine that is powering a range of socially aware applications. At the time of writing, Elgg had been downloaded around 500,000 times with over 900 plugins contributed; prompting more than two million downloads. As the community becomes ever more involved, I feel the future is bright for Elgg.

Dave Tosh
Elgg Co-Founder

About the Author

Cash Costello performs research for the Johns Hopkins University Applied Physics Laboratory. In addition to his work there in computer vision and machine learning, he coordinates a team of developers in the creation of collaborative web applications.

Cash is also a core developer of the Elgg Social Networking framework. He is active within the Elgg community, whether sharing his plugins or helping others to get the most out of Elgg.

Thank you to all the editors from Packt that were involved in shepherding this book from concept to publication. I would also like to thank all the reviewers who contributed so much through their feedback. This book would not have been possible without the original work of Ben, Dave, and Marcus in creating Elgg. The version of Elgg described in this book is the result of a lot effort by Brett and Evan. Great job, guys.

Most of all I would like thank my wife for her encouragement and patience during this very long process. She read every page regardless of interest in web development. Merci beaucoup, mon amie.

About the Author of 1st edition

Mayank Sharma is a contributing editor at SourceForge, Inc's Linux.com. He also writes a monthly column for Packt Publishing. Mayank has contributed several technical articles to the IBM developerWorks where he hosts a Linux Security blog. When not writing, he teaches courses on Open Source topics at the Indian Institute of Technology, Delhi, as guest lecturer.

Thanks to my mom Shashi and dad Rakesh for laying down tough standards, and younger brother Shashank who beat me at book-writing to become my mentor.

Gratitude to David Barnes for guiding me from start to finish, to Rashmi Phadnis for her edits, to Patricia Weir for managing and scheduling the book, and to the Elgg developers and community for such a wonderful project.

About the Reviewers

Valentin Crettaz holds a master degree in Information and Computer Sciences from the Swiss Federal Institute of Technology in Lausanne, Switzerland (EPFL). After he finished studying in 2000, Valentin worked as a software engineer with SRI International in the Silicon Valley (Menlo Park, USA) and as a principal engineer in the Software Engineering Laboratory at EPFL. In 2002, as a good patriot, he came back to Switzerland to co-create a start-up called Condris Technologies, a new venture that provides IT development and consulting services and specializes in the creation of innovative next-generation software architecture solutions as well as secure wireless telecommunication infrastructures.

From 2004 to 2008, Valentin served as a senior IT consultant in one of the largest private banks in Switzerland, where he worked on next-generation e-banking platforms.

Starting in 2008, Valentin joined Goomzee Corporation as Chief Software Guru. Goomzee is a Montana based mobile marketing company that provides solutions for connecting buyers and sellers in any market vertical through mobile interactions.

Valentin also owns a small consultancy business called Consulthys, a new venture that strongly focuses on leveraging Web 2.0 technologies in order to reduce the cultural gap between IT and business people.

Marcus Povey is a software architect with a wide range of commercial experience, including portable medical systems, point-of-sale hardware, web platform development, and secure messaging.

Formerly a Senior Architect at Curverider, Marcus worked with Ben Werdmuller to develop the open source social networking platform Elgg, and was an integral part of its architecture design process from version 1.0 onwards.

Marcus left Curverider in 2009 to form his own consultancy company; he also organizes BarCamp Transparency, an annual event to discuss openness and direct democracy in government.

Marcus maintains a blog at `http://www.marcus-povey.co.uk`

Brett Profitt has been interested in computers and programming since his youth. Whether it was pecking at keys on a Commodore 64 until a monochrome display echoed his name, leading a middle school computer club, or writing apps on his TI-83 calculator during chemistry class, computers have always been an important part of his life as one of his passions.

Brett received two degrees from The Ohio State University: A Bachelor of Instrumental Music Education and a Bachelor of Art in Japanese Language and Culture. Before adopting software development as a career, he held positions as a pre-school teacher, website designer, Kindergarten music teacher, and tutor for Japanese-speaking students in America.

Brett strongly believes in and supports open source philosophies. He is proud to be a member of this powerful community and enjoys interacting and co-developing with the larger open source community through Elgg.

I would like to thank my family who have provided inspiration and support in all my pursuits, no matter if arts, sciences, or languages. I would also like to thank my friends, who make Friendsgiving the best holiday of the year.

Liran Tal is a leading software developer, expert Linux engineer, and an avid supporter of the open source movement. In 2007, he has redefined network RADIUS management by establishing daloRADIUS, a world-recognized and industry-leading open source project. Passionate about creating software and enjoys taking on new ventures, he is mostly focused on building web applications and social networking technologies.

He graduated cum laude in his Bachelor of Business and Information Systems Analysis studies and enjoys spending his time playing the guitar, hacking all things Linux, and continuously experimenting and contributing to open source projects.

Sincere thanks to Curverider for creating Elgg and making it available and to everyone on the Elgg community who has been there to push it forward.

I'd like to thank Tal, my wife, for her love, support and patience; and my mom and dad, for being a source of inspiration in my life.

Evan Winslow holds a B.S. in Computer Science from Stanford University. He has been working with Elgg since 2009 and has been a member of the Core Development Team since 2010. As a member of the Core Team, he contributed significantly to the JavaScript and CSS advances in Elgg 1.8. Evan lives in Aliso Viejo, California with his wife Julie and son James, and works his dream job doing front-end web development full-time. You can reach him at `evan@elgg.org`.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit www.PacktPub.com for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at www.PacktPub.com and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at service@packtpub.com for more details.

At www.PacktPub.com, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

http://PacktLib.PacktPub.com

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at www.PacktPub.com, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface

Elgg is a web application for social networking. It has all the features you would expect from a social web application. It has blogging, file sharing, social bookmarking, microblogging, activity streams, groups, "friending", user profiles, and the list goes on. However, Elgg is more than just a web application. It can also be used as a development framework for creating social websites. Developers are building impressive social media sites on top of the Elgg engine through its powerful API. They are using it to add social functionality to current websites and integrating it with other popular web applications.

Elgg is open source, licensed under the GNU General Public License (GPL). You can download, install, and use it without cost. Taking advantage of its plugin architecture, there is a community of users and developers contributing plugins and themes for others to use. Open source software and an open source community are a great combination that everyone can benefit from.

Anyone can use Elgg to create a customized social networking site. Entrepreneurs are building specialized social networking sites with it. Educators are using it as an e-learning tool. Corporations are adding it to their intranets to better connect their employees. There is a wide range of applications for Elgg and with it you have complete control over your site and your data.

This book has two main objectives: help you understand what functionality Elgg provides and explain how you can customize it to make it do exactly what you want. It is not a manual for administering an active Elgg-based site, nor is it a manual for users of Elgg sites. This book is all about using and customizing Elgg to build a social website.

What this book covers

Chapter 1, *Social Networking and Elgg*: This chapter describes the features that drive today's social networking and social media websites. It provides an overview of Elgg along with a list of web resources focused on Elgg and its users. Also included is a discussion of common uses for Elgg that go beyond the typical Facebook-like social networking site.

Chapter 2, *Installing Elgg*: Before you can start using Elgg, you need to install it. This chapter guides you through the process of setting up Elgg.

Chapter 3, *A Tour of Your First Elgg Site*: Nothing tells you more about software than using it, and this chapter starts you on a hands-on exploration of Elgg's capabilities. Topics covered include creating user accounts, setting up user profiles, and administration.

Chapter 4, *Sharing Content*: Once you have your Elgg site up and running, you will learn how to share content with Elgg. Blogs, bookmarks, files, and more can all be shared using the core plugins that come with Elgg.

Chapter 5, *Communities, Collaboration, and Conversation*: This chapter shows you how to use the group's capability to create virtual communities on your Elgg site. It continues by describing the different tools available for users to communicate with each other.

Chapter 6, *Finding and Using Plugins*: After reading about all the features that Elgg has out of the box, you now get to extend it with plugins created by members of the Elgg community. The chapter has an overview of installing, configuring, and testing plugins followed by a detailed look at three major community plugins.

Chapter 7, *Creating Your First Plugin*: This chapter follows step-by-step as we create a "hello world" plugin and introduce many components of Elgg's plugin API. This chapter also offers advice on debugging a plugin when your code is not working like you want it to.

Chapter 8, *Customization Through Plugins*: Learn how to customize Elgg through creating your own plugins. This chapter is organized as nine lessons that teach you different aspects of writing plugins.

Chapter 9, *Theming Elgg*: One of the best ways to impress potential users is through the visual design of your site. This chapter describes how themes work and how to create your own. To get the most out of it, you will need basic knowledge of HTML and CSS.

Chapter 10, Moving to Production: Everything that you have done with your site so far has been to experiment and understand Elgg, but now you are thinking about opening it to the public. What sort of server do you need? How do you back up the data? What do you do when the spammers find you? This chapter addresses these kinds of questions.

Appendix A, Developer's Quick Start Guide: Elgg is a powerful development platform. It was designed for extensibility, and developers can be very productive building on it with a solid understanding of how Elgg works. This appendix provides an overview of Elgg as a development platform. It gives you a big picture view of how it works before you start writing code.

Appendix B, Views Catalog: This appendix is a visual catalog of Elgg's views. Along with the description and picture of the view, it includes hints for developers and themers on their uses.

What you need for this book

If you have a web hosting service that supports PHP, then you can install Elgg and start exploring its capabilities. If you do not, then you can install a package like XAMPP on any computer so that you can follow along as we use and customize Elgg. Complete information on installing Elgg is available in *Chapter 2*.

Who this book is for

This book is written for people interested in using Elgg to build a social networking or social media website. You may be evaluating Elgg for a possible project, in the middle of using Elgg to create a site, or simply checking out open source web applications for future use.

While the intended audience includes web developers, it is not written exclusively or even primarily for developers. Those who build sites based on Elgg come from a wide range of backgrounds and many do not have software development experience. You could be an educator, entrepreneur, scientist, student, or waitress. We do not assume that you have experience working with code, but certainly even a little experience will help you as you work through the material in this book.

Elgg can be customized through configuring options and installing plugins. If you want more control over Elgg, then it will require working with code. You will not need years of experience in web development to benefit from the chapters on writing plugins, but they will challenge those who are new to the PHP language and web development.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: " Next, we pass the content into `elgg_view_layout()` through the associative array `$vars`."

A block of code is set as follows:

```
<?php
/**
 * Hello world plugin
 */

  elgg_register_event_handler('init', 'system',
    'hello_world_init');

  function hello_world_init() {
  // do nothing right now
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Enter your settings and click on **Save**".

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a book that you need and would like to see us publish, please send us a note in the **SUGGEST A TITLE** form on `www.packtpub.com` or e-mail `suggest@packtpub.com`.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/support`, selecting your book, clicking on the **errata submission form** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at copyright@packtpub.com with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at questions@packtpub.com if you are having a problem with any aspect of the book, and we will do our best to address it.

1 Social Networking and Elgg

The Web has become social. Is today your birthday? Your family and friends are posting best wishes right now on Facebook. Are you planning to go out for dinner? You are likely checking user reviews of restaurants on a website such as Yelp. Did you read that news article about the latest political scandal? If you did, you may have discussed it with other readers in the comment section of the online news site. The Web has always been a great place to find information, but increasingly people are also using it for social interaction.

With this shift has come a new class of web applications focused on social interactions like those described above. People are sharing and commenting on photos and videos using sites such as Flickr and YouTube. They are updating their "followers" on Twitter about their latest activities. They are using social networking applications such as Facebook to reconnect with old friends, share information about their lives with friends and family, or find people with common interests.

The rich interactions available on these social sites have raised the expectations of users for web-based applications. It is not enough to search for and view interesting content. People want to interact with it, comment on it, and discuss it with others. Incorporating social features into a website leads to users spending more time on the site.

The growth in usage of social functionality and social networking in particular, is not limited to consumer websites. Many companies are using social networking software on their corporate networks for internal collaboration. Employees can use these tools to find expertise that exists within the organization and form groups that promote sharing information across organizational boundaries. In education, social networking applications are used to help students actively collaborate on projects. Students can work together, share ideas, and discuss each other's work throughout the learning process.

Social features

As you have used social sites such as Facebook, YouTube, or Twitter, you may have noticed features common to many of them. These features distinguish this new generation of social web applications from the less interactive websites that came before them.

Profiles

A profile lets users describe who they are. It can be as simple as a name and a few vital statistics such as location and age or as elaborate as a complete listing of a user's background, likes, dislikes, and interests. A profile usually includes a photo (often called an avatar) that represents the user throughout the site.

Relationships

Social sites often let you "friend" or "follow" other users. These relationships define who is in your social network and are often used to control access to content. For example, a user may set the permissions on a photo so that only friends may see it.

Content sharing

Many social sites focus on sharing content; think Flickr, YouTube, or Blogger. People not only share photos, videos, blogs, and files, but they also comment, rate, and recommend.

Activity and notifications

With all this sharing and commenting, people need a way to keep track of what is important to them. An activity stream that displays the latest activity relevant to a user is one technique. A good example of this is the news feed in Facebook. E-mail notifications are another common method for staying updated on what is happening.

Groups

Not only do people form relationships with other users, but they also join groups organized around shared interests or purposes. These groups often mirror the communities that people join outside of the virtual world of the web: alumni groups, sports fans, book clubs, and charities. Social websites enable these groups to form and thrive regardless of the location of the members.

Communication

Social interaction on these websites is not limited to commenting and sharing content. Encouraging communication between users is another important attribute—whether in public such as a message board or discussion forum or in private with real-time chat or e-mail-like messaging.

These are the types of features that are expected in today's websites. This expectation has created a demand for frameworks and web applications that provide social functionality. Solutions range from hosted social networking sites from providers such as Ning, to sets of plugins that extend content management systems such as Drupal and WordPress, to full application frameworks focused on social networking such as Elgg. The end result is that it is easier than ever to create custom social networking websites.

What is Elgg?

Elgg is an open source platform for building social websites, especially social networking sites. Now that sentence may require some parsing, so let's start with the fact that it is free open source software (FOSS). This means you are free to use it however you want, free to modify it, and free to redistribute it. Its development is managed by the nonprofit Elgg Foundation and developers from all over the world contribute code to make Elgg better.

Elgg is used to build social websites. It has social networking in its DNA and provides all the social features mentioned in the preceding section. Think of the buzzwords that you associate with social media: blogging, sharing, tagging, friending, or tweeting. Elgg does all of that.

Elgg is a platform that provides the building blocks for creating great social websites. While you can install Elgg and immediately begin using it as a social networking site, most of you will want to customize it. Those of you who are not developers can download plugins to change the look and feel or add new capabilities. For the developers, almost every part of Elgg can be customized by writing new plugins.

The recommended server configuration for running Elgg is called a LAMP stack. This stands for Linux, Apache, MySQL, and PHP and is the most common hosting environment on the Web. Linux is the operating system, Apache the web server, MySQL the database, and PHP is the server scripting language. Elgg can also run on other operating systems such as Windows or Mac OS X and with other web servers.

With some basic experience setting up web applications, you can go from downloading Elgg to having a functioning social networking site in 10 minutes. After it is installed, you will spend time evaluating it, testing different plugins and themes, designing your site, and building new plugins (if you are a developer). This book serves as your guide to the process of creating a site like one of those shown in the following screenshot:

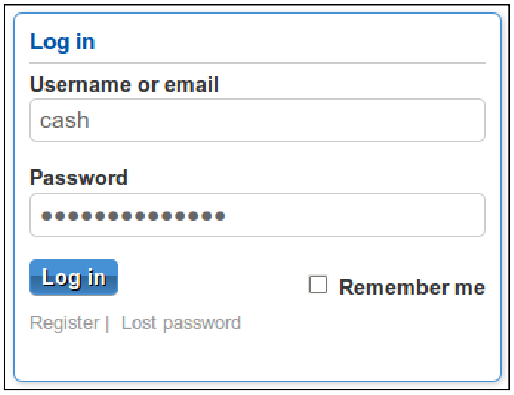
LIVE OUT THERE
CLUB RED Admitted Student Network
RUCKU
Rugby's virtual clubhouse
community@Brighton
elgg
elggCommunity
farmsphere
the food & agribusiness meeting point
Handicap
HEDGEHOGS
rézo

The Elgg engine

The Elgg platform is divided into two parts: a core engine and plugins that extend that engine. The engine contains the basic building blocks needed for a social website. It also provides the framework for developers to create new social tools through plugins. Here is a quick overview of what the Elgg engine does for your site.

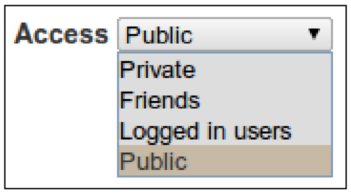
User management

The Elgg engine handles basic user account creation and management. Registration, logging in, password resets, and e-mail address changes are all handled for you by the engine. Elgg also supports external authentication so that sites can integrate user accounts with other services. For example, there are plugins available that use this capability to enable users to log in with their credentials from Facebook or Twitter.



Privacy controls

Users want to control who has access to their data and the Elgg engine does this through a granular permission system. Every piece of content in Elgg has an access level assigned to it. It could be a blog post, a video, or an individual profile element. The engine gives users tremendous flexibility in who can see their data through simple privacy controls and custom access lists.

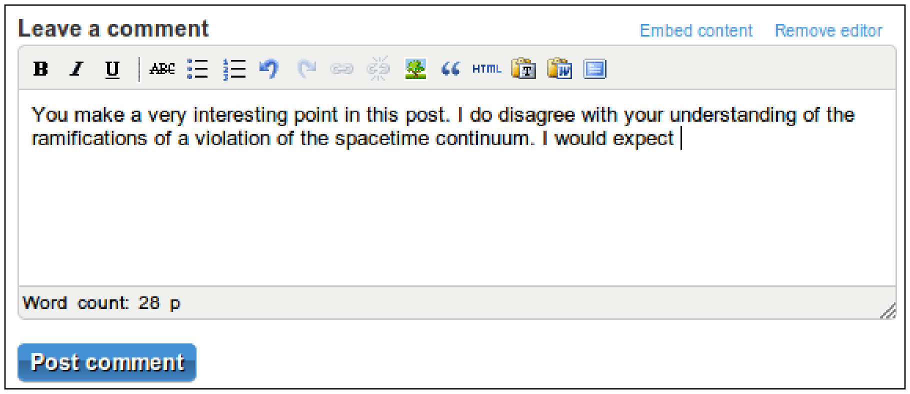


Theming

The engine has all the hooks so that you can do anything from tweaking the default theme to writing a completely new one. You have control over every line of HTML, every layout, every CSS statement. *Chapter 9*, *Theming Elgg* describes how to theme Elgg.

Commenting

It is no fun to share your latest vacation photos without people commenting on them. Elgg has a built-in commenting system that can be applied to any content in the system, whether blogs, bookmarks, or videos. This is great for plugin authors because they do not have to write any code to handle comments. It is also good for you as the commenting system looks and works the same way throughout the site.



You make a very interesting point in this post. I do disagree with your understanding of the ramifications of a violation of the spacetime continuum. I would expect

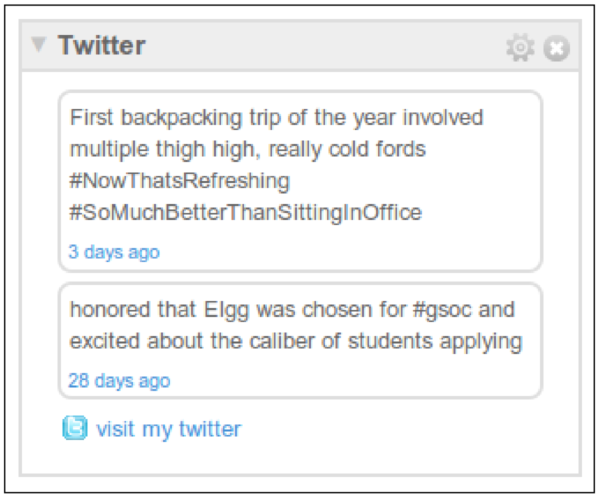
Word count: 28 p

Tagging

A feature common to social sites is tagging. Tags are keywords attached to content. When you click on a tag, it shows you all the content that shares that keyword. Elgg has a common interface for tagging content—whether blog posts, files, or a user's profile. The engine also makes it easy to create tag clouds.

Widgets

Widgets are draggable, customizable components that are often found on web portals such as iGoogle or My Yahoo!. Many of the plugins that are distributed with Elgg have their own widgets that can be placed on a user's profile page. The widgets display members' latest blog posts, who they are friends with, or what groups they have joined. Most social media sites provide bits of JavaScript that can be used to create new widgets within Elgg. For example, there is a widget that displays a user's latest tweets from Twitter.



First backpacking trip of the year involved multiple thigh high, really cold fords #NowThatsRefreshing #SoMuchBetterThanSittingInOffice

3 days ago

honored that Elgg was chosen for #gsoc and excited about the caliber of students applying

28 days ago

Internationalization

Your users may not all speak the same language and you might want your site to support their native languages. The Elgg engine can load different language files based on user preference. By default, Elgg's engine and the bundled plugins come with English language files. People in the Elgg community have created translations for languages ranging from German to Chinese to Basque.

Feeds

A challenge for any social site is how to keep people coming back. E-mail notification is one popular technique that Elgg supports. Another option is providing RSS feeds so that users can monitor activity on your site. The Elgg engine can turn almost any page into a RSS feed. Do you want site-wide activity? There is a feed for that. How about the latest posts in a group forum? There is a feed for that. Looking to stay updated on the newest comments on a blog post? There is a feed for that, too. Just look for the orange feed icon at the top of the sidebar menu or in your web browser's location bar.

Web services

A majority of Twitter's traffic comes from desktop and mobile clients. How does this work? Twitter provides a web services API so developers can write applications such as TweetDeck, which interfaces with Twitter. The applications can pull the latest tweets out of Twitter and can post new tweets for the user. Elgg has a framework for building web services APIs much like those of Twitter or Flickr. Think of all the possibilities: desktop clients for notifications, mobile clients for sharing photos, mashups that use your site's data.

The power of plugins

We just reviewed part of the functionality of the Elgg engine, but you probably noticed that there was no mention of blogging, status updates, or other features common to social networking sites. That is because these features are offered through plugins. The plugins modify or extend Elgg giving you control over what features are included in your site.

Bundled plugins

Elgg is distributed with a set of plugins which are written and supported by the Elgg development team. These are called the bundled plugins. These plugins provide enough functionality to run a basic social networking site and are a good demonstration of what is possible using Elgg's plugin system.

The bundled plugins provide a wide range of functionality. Included are content sharing plugins such as blogging, social bookmarking, and file sharing. There are plugins for user profiles, activity streams, groups, and notifications. Integration with Twitter, private messaging, search, and administrative tools are also provided through these plugins. Detailed information on the bundled plugins is included in *Chapter 3*, *A Tour of Your First Elgg Site*, *Chapter 4*, *Sharing Content*, and *Chapter 5*, *Communities, Collaboration, and Conversation*.

Third-party plugins

As an open source project, Elgg has a community of developers who build plugins for their own use and then share them with other Elgg users. These plugins are referred to as third-party plugins and many are available from the plugin repository on the Elgg website (`http://community.elgg.org/pg/plugins`). Some of these plugins add significant functionality to Elgg such as a photo gallery or event calendar. Others extend a bundled plugin or customize an aspect of the Elgg engine. There are hundreds of third-party plugins available. *Chapter 6*, *Finding and Using Plugins* describes a few of the best and gives advice on how to select and test any plugin.

Themes

The look and feel of Elgg is controlled by the theme. Elgg comes with a default theme that you can use. It is not a separate module that can be uninstalled, but is built into the engine. To change the theme, you install a theme plugin that overrides parts or the entire default theme. Theme plugins are installed just like other plugins and have access to the same APIs. *Chapter 9* provides information on where to find themes for download, describes the components of a theme, and includes a guide to creating your own.

Building your own plugins

You can build your own plugins to customize or extend Elgg. Writing a simple plugin requires either basic knowledge of (or the motivation and persistence to learn) both PHP and HTML. An example of a simple plugin is one that changes the word "blog" to "report" in the blog plugin's menus and page titles. Creating a plugin that does this is quite easy to write. There is a tutorial in *Chapter 8, Customization Through Plugins* that shows how this can be done.

A more complicated plugin is one that modifies how Elgg handles the creation of "friend" relationships. Elgg's default model is that any member can friend any other member without requiring a confirmation. A one-way relationship is formed, meaning that the fact that I am your friend does not imply that you are my friend. This is consistent with Twitter's model of followers. For your social networking application, you may want two-way relationships that require confirmation as Facebook does. The plugin would hook into Elgg's engine and change the relationship creation process. A reciprocal friendship plugin like this is available in the Elgg community plugin repository.

With the right development skills, you could also write a plugin that adds a significant new capability to Elgg or integrates Elgg with another open source web application. Elgg's plugin API is quite powerful so the only limitations to what you can build are your skills, time, and creativity.

Case studies

To get the creative juices flowing and as a demonstration of how Elgg can be used, consider the following three case studies.

Niche social network

James wants to create a social networking site for tango dancers around the world. He wants his users to be able to upload tango music and playlists. They can upload videos of tango dancing and rate them. Each city will have its own group where members can discuss tango related activities. The groups should have a calendar that lists the upcoming dances and a place for people to post photos of previous events. James also wants to pull in posts by tango bloggers from outside his site and redistribute them as a feed to make it easy for people to follow what people are saying in the community. He, of course, also wants a slick theme that elicits the emotion of the dance.

Designing and building the site

James has selected Elgg to run his site. As he looks through what plugins are available, he sees that there is already a file sharing plugin with an extension for playing MP3s in the page. He decides to rename the default blog plugin to playlists so that people can create and comment on them. James is delighted when he sees a video plugin that uses the Kaltura video site as the backend. This means the videos will not be stored on his server, decreasing his storage requirements.

For the city-focused groups, he uses the Elgg group's plugin in combination with plugins that provide an event calendar and a photo gallery. He likes that the gallery plugin provides photo tagging and hopes that the tagging notifications gives people a reason to keep coming back to the site. He still needs to figure out how to pull in the blog feeds of tango bloggers. He finds a plugin that displays RSS feeds on user profile pages, but it requires additional development to make it do exactly what he wants. He happens to have a friend with PHP experience and with a little free development help; he now has an aggregated tango blog feed.

With theming he has two choices: either download a free theme and modify it to fit his needs or contract with a web designer to build one for him. James decides that the visual interface is too important to skimp on and uses a freelance designer.

Deploying the site

The only step left is convincing people to use his site. James has been smart. By resyndicating the blog feeds he has won the favor of the bloggers who now link back to his site. He also recruited a few well known dancers from key cities to join. James's final step is to contract a developer to create an invitational system plugin. Each member gets a limited number of invites and the only way to join is to get an invite. This helps to create a buzz about the new tango site.

From this example, you not only see how James used Elgg but also that creating and growing a social networking site is more than software. A common mistake is spending a lot of time building a site and forgetting about the challenges of getting people to use it.

Plugins mentioned

Four of the plugins mentioned are distributed with Elgg: file sharing, MP3 player (zaudio plugin), blogging, and groups. The Kaltura collaborative video plugin by Ivan Vergés, event calendar plugin by Kevin Jardine, the Tidypics photo gallery plugin by the Tidypics team, and the Simplepie RSS feed plugin by Cash Costello are all available from the plugin repository on the Elgg website.

Corporate intranet

Imagine a theoretical company called Acme Corporation. Acme has deployed applications on their intranet to encourage collaboration among their employees. They are using MediaWiki as their wiki and Sharepoint for file sharing. Management and the employees are not satisfied with the current solution. They still feel that it is difficult to find the right person with the right expertise for a project, quickly organize ad hoc teams, or form communities of practice that cut across organizational boundaries. They need software to tie together the current intranet applications while providing the social functionality needed to address the mentioned limitations.

Designing and building the site

The collaborative software team decides to create a prototype with Elgg to evaluate a social networking solution. The first requirement is that the users need to be able to log in using their Active Directory credentials. Fortunately, there is already a LDAP plugin available and with a small amount of configuration, it is up and running. The second requirement is supporting the formation of communities of practice. The group plugin distributed with Elgg provides this capability and it is activated on the development server.

Next, on the list is user profiles to help with the challenge of finding expertise. The profile plugin that comes with Elgg has tagged profile fields, but the development team felt they needed a more powerful and extensible profile capability. A third-party plugin that extends the profile plugin to provide profile types and more customizable profile fields was found on the Elgg community plugin repository.

Integration with MediaWiki and Sharepoint are the remaining requirements. Another organization released a MediaWiki plugin that pulls content out of the wiki and makes it available within Elgg. That was a quick solution, but integration with Sharepoint proves to be more difficult. After some research, they decide to write a custom plugin to use Sharepoint's web services API. The developers end up tying in updates on files in Sharepoint to the activity stream provided by Elgg. Now users can get updates on anything that is happening in the wiki, Sharepoint, or Elgg in one place.

A few of the developers have been using Twitter so they try out Elgg's microblogging plugin. The team likes its ease of use and the ability to share quick status updates, so they decide to include it in the production system.

Deploying the site

Before making the site available, the development teams decides to create accounts for all the employees first. They write a plugin that extends the LDAP plugin that grabs all the users from the Active Directory server, creates the accounts, and fills in some of the profile fields. To speed adoption of the new application, particular communities that would benefit from the groups capability are targeted. Very soon there is an active group for Java developers with shared code snippets and discussion threads on unit testing, distributed versioning systems, and many other topics.

Plugins mentioned

Core plugins mentioned were groups, the wire (microblogging), and profile. The profile manager plugin by ColdTrick IT Solutions is available from the Elgg community plugin repository. The MediaWiki integration plugin written by the MITRE Corporation is also available there. The LDAP plugin is available from Elgg's github account (`https://github.com/Elgg`).

Educational collaboration

Mr. Harris teaches middle school history and wants to use collaborative technology for a group project idea that he has. He needs a web-based application that is simple to set up and requires minimal configuration. Mr. Harris wants the students to break up into teams, research a historical figure, and then create an online profile for that person. The profile will include biographical information along with status updates and blogs written from the perspective of the historical figure. He also wants the students to use the site to collect resources and collaborate on the writing process.

He decides the tools he needs are blogging, groups, bookmarking, and collaborative document editing. Another requirement is that the students need to be able to access it at home and school, but no one else on the Internet should be able to see the data. One last requirement is that it must be cheap—preferably free.

Designing and building the site

Mr. Harris was recently at a conference on online collaboration in the classroom and remembers Elgg being mentioned. As he looks through the plugins, he sees all the tools that he needs. Blogging, groups, profiles, status updates, and social bookmarking are all available.

The teams will use the social bookmarking plugin to save and share online resources. The pages plugin will be used by the teams to collect the information they have gathered, both Internet and book based, in one place before they create the profile or write the blog posts. Mr. Harris likes that the pages plugin records who has been working on each web page so that he can see who is contributing to it.

The last piece is making the site a walled garden so that only students and school staff can log in and see the data. Fortunately, Elgg got its start in the educational arena and has that feature built-in. Because Elgg is free open source software, Mr. Harris was able to do all of this without any cost other than his time.

Deploying the site

The school IT staff installs Elgg on a server. Mr. Harris creates a spreadsheet with all the students' information and uses an importer plugin to create accounts for them. The experiment is a success. The students like working in an environment that feels like Facebook and are comfortable using these types of tools. A few of the blog posts that they wrote are quite entertaining.

Plugins mentioned

The pages, groups, blog, bookmarks, profile, and the wire (for user status) plugins are all bundled plugins distributed with Elgg. The user importer plugin is available from the Elgg github account.

Elgg resources

There are many resources available on the Web to help you use Elgg. Most of these live on the `elgg.org` domain that is run by the Elgg Foundation.

Elgg community

The Elgg community site (`http://community.elgg.org`) provides a place for Elgg users and developers to interact. There are group forums where people can ask questions and share expertise. The plugin repository at the community site has hundreds of plugins that have been contributed by developers.

Elgg wiki

A great place to go for documentation on Elgg is the wiki at `http://docs.elgg.org`. It covers the installation process, administering an Elgg site, and creating plugins using Elgg's API. The wiki also contains a list of sites based on Elgg which can provide inspiration on what is possible with Elgg. Remember that it is a wiki so you can add to it.

Developer resources

The Elgg project uses Trac for bug tracking and managing release milestones (`http://trac.elgg.org`). The latest development version of the code is found on Github (`http://github.com/Elgg`). For Elgg developer community communication, try out the Google group (`http://groups.google.com/group/elgg-development`) and the Elgg IRC channel: #elgg on Freenode. To keep up with the latest efforts of the Elgg development team, follow the blog at `http://blog.elgg.org` and the @elgg Twitter account.

A few words of advice

Throughout the book there is practical advice gained from experience building, developing, maintaining, and administering Elgg sites. Before you begin installing Elgg in the next chapter, take a little bit of time to read the general, but important advice below.

Take notes

As you work through the next few chapters, you will notice things that you would like to change or add. Jot those down in your notes so that when you are done, you have a list of possible customizations to prioritize.

Save resources

As you try out Elgg, you are going to be using the resources listed above. You will find interesting tutorials or hints about Elgg that you won't need to use at that moment. Save links to those resources—whether through browser bookmarks or web-based sites such as Delicious. In addition, as you browse the Web, you are going to run across features or design elements that you really like on other sites. Save those, so they can serve as inspiration as you design your site.

Be methodical

It is tempting to go to the Elgg community site, download 10 or 20 plugins, and install all of them at once. This is not a good idea. If one of the plugins causes a problem with your site, you will not know which the bad one is. It is better to systematically install and test one plugin at a time. *Chapter 6* walks you through the process of finding, installing, and configuring plugins.

Finding help

Whether looking for guidance on how to do something or help fixing a problem, the best first stop is the Elgg wiki. Do a search there or check out the FAQs. If you do not find your answer, visit the Elgg community site. Search the forums to see if anyone has asked the same question before. If not, select the group that best matches your issue (often the Technical Support group) and ask your question. Be sure to use a descriptive title and provide plenty of details.

Using Google search with the Elgg community site

The search built into Elgg is not bad, but is not nearly as powerful as Google's search engine. You can search the Elgg community forums by including `site:community.elgg.org` in your Google search query.

Summary

The growth of social networking and other social media sites will be a continuing trend. Elgg is a powerful open source platform for building these types of websites. The combination of the core social engine and the customization offered through plugins provide the capabilities and extensibility needed for today's social websites. This chapter was a quick introduction to what is possible with Elgg. The following chapters give you hands-on experience with Elgg by guiding you through the processing of installing and using this software.

2
Installing Elgg

It is time to create your first Elgg site. This will be a test site that you will use throughout the book as we explore what Elgg can do. If you have installed web applications such as Elgg before, the installation should only take a few minutes. If you are new to this, it will take a little longer as you work through each step with the assistance of this chapter.

To install Elgg, you need to:

1. Check that your server meets Elgg's requirements.
2. Download Elgg and put its code in the server's web directory.
3. Create a data directory and set its permissions.
4. Create a database for Elgg.
5. Run the web-based installer.

The instructions in this chapter work for a typical server. If there are complications while performing the installation, use the Elgg wiki as a resource (`docs.elgg.org`) for assistance. It has detailed information about server requirements, installing on particular types of servers, and troubleshooting.

Preparing the server

Elgg is a server application, so you need a server to run it. Your server options fall into three categories:

- **Local server**: If you are at school or the office, there may be a server already configured with a web server that you can use for Elgg.

- **Web hosting provider**: A popular option is renting space on a server. The hosting company handles the maintenance of the server, provides technical support, and may even provide a one-click installer for Elgg. *Chapter 10, Moving to Production* includes a section on selecting a web hosting package when preparing to launch a live website.
- **Home computer**: You can use your desktop or laptop computer, whether it runs Windows, Mac, or Linux. There are packages available for download, like XAMPP, that include everything you need for running Elgg. This is a great way to test out Elgg before you select a server for your site.

Platform specific instructions

There are detailed, step-by-step installation instructions specific to your operating system on the Elgg wiki (`http://docs.elgg.org/Installation`). They are a great resource for those who are new to installing web applications.

Checking requirements

Now that you have identified a server, you need to confirm that it meets Elgg's requirements (almost all servers meet these requirements):

- **PHP scripting language version 5.2 or greater**: Many hosting services will list their PHP as version 5, but it is usually 5.2 or 5.3.
- **Web server software that supports PHP and URL rewriting**: For most people this will be the Apache web server. The installer will check if URL rewriting is working. (URL rewriting enables you to have short, readable URLs such as `http://elgg.org/profile/cash` rather than long confusing ones such as `http://elgg.org/engine/handlers/page.php?type=profile&user=cash`.)
- **MySQL database version 5 or greater**: Elgg does not work with other databases.

If you are not sure that your server meets these requirements, contact the technical support of your hosting company or the administrator of your server. Additionally, Elgg's web-based installer checks the requirements when you run it.

Downloading the latest version

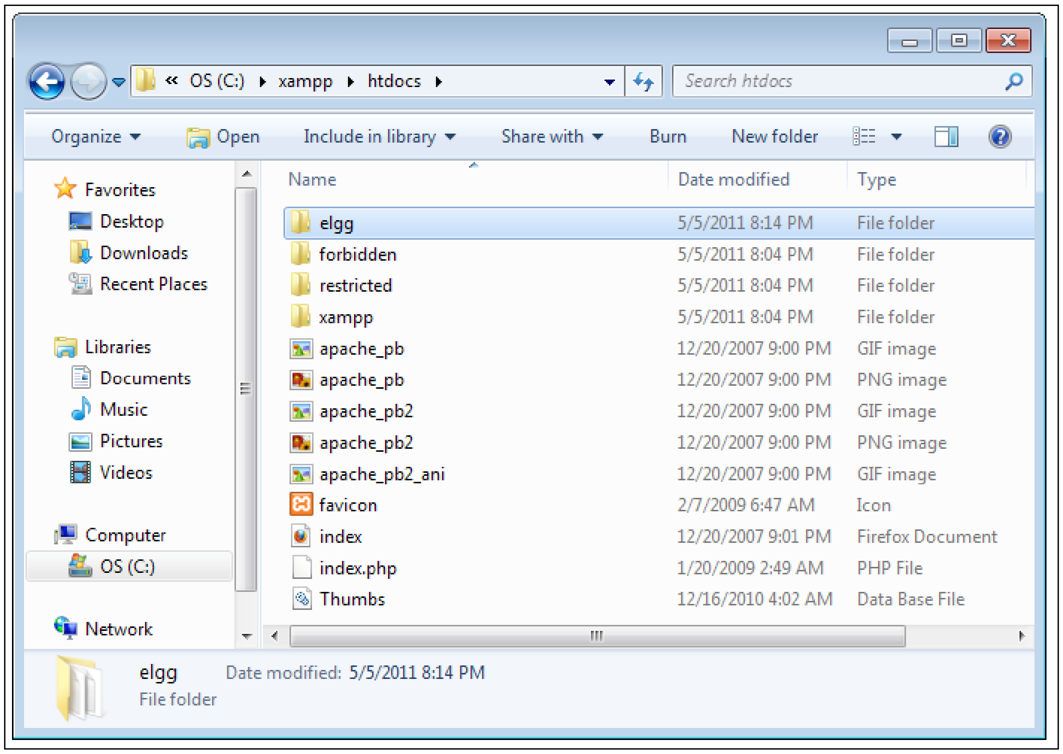
The latest release of Elgg is available at the `elgg.org` website. When you go to the site, look for the large download button. Clicking that button takes you to the download page. You want to download the latest recommended release. The ZIP file will be around 3 MB in size.

Extracting the files

Before moving the files to your server, you need to extract them from the archive. If you are running a recent version of Windows or Mac OS X, you can double-click the ZIP file to open it. Then, you can drag-and-drop the Elgg folder to wherever you want to temporarily store the files. The location does not matter as long as you can find it again.

Moving files to your web server directory

If you are using your own computer as a test server, you must copy the files into Apache's web directory. For XAMPP, it could look like this:



Name	Date modified	Type
elgg	5/5/2011 8:14 PM	File folder
forbidden	5/5/2011 8:04 PM	File folder
restricted	5/5/2011 8:04 PM	File folder
xampp	5/5/2011 8:04 PM	File folder
apache_pb	12/20/2007 9:00 PM	GIF image
apache_pb	12/20/2007 9:00 PM	PNG image
apache_pb2	12/20/2007 9:00 PM	GIF image
apache_pb2	12/20/2007 9:00 PM	PNG image
apache_pb2_ani	12/20/2007 9:00 PM	GIF image
favicon	2/7/2009 6:47 AM	Icon
index	12/20/2007 9:01 PM	Firefox Document
index.php	1/20/2009 2:49 AM	PHP File
Thumbs	12/16/2010 4:02 AM	Data Base File

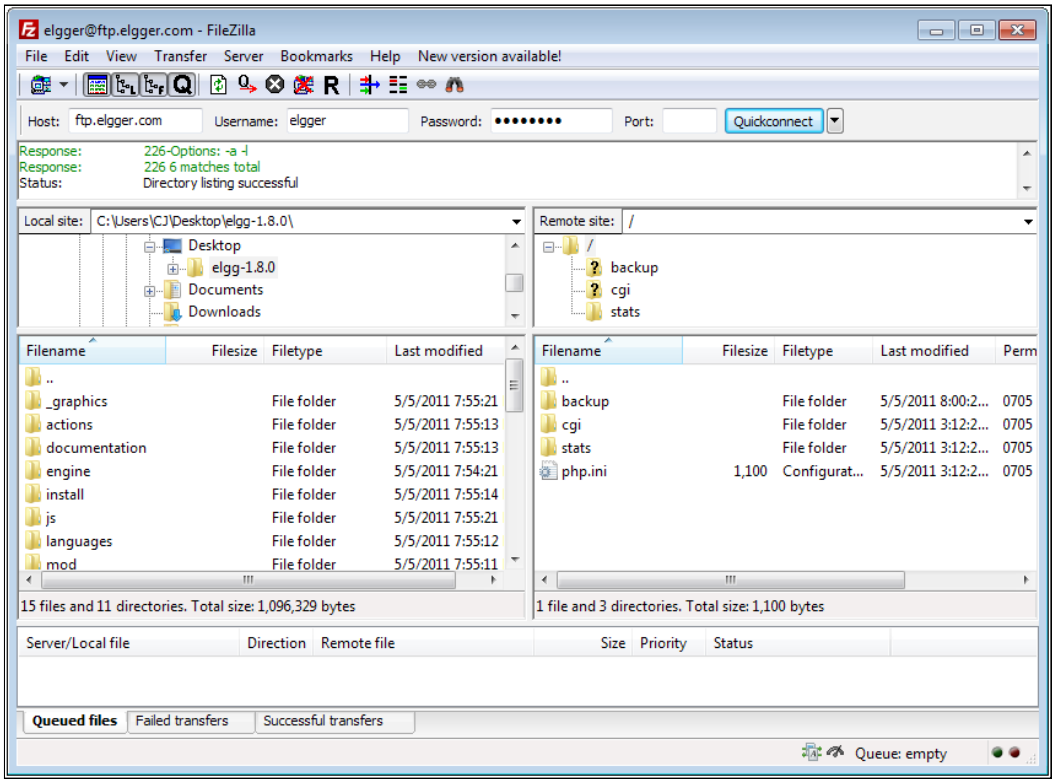
elgg Date modified: 5/5/2011 8:14 PM
File folder

If you are using a remote server, you need to FTP the files to the server. You will need to know the username and password for your server account and will need an FTP client. A great FTP client that runs on all the major operating systems is FileZilla (`http://filezilla-project.org/`).

Start your FTP client. Your hosting provider should have given you information on how to connect to your server including the address. Enter this information and connect to the remote server. After you are connected, you should see file explorers for both your local computer and the remote server. On the local side, navigate to where you put the Elgg files and on the remote side, navigate to your web directory.

If you want Elgg in a subdirectory, copy the folder named `elgg-1.8.0` to the remote server. You can then rename the folder to get the address that you want. For example, renaming the folder `social` would give you an address like `http://example.org/social/`.

If you want Elgg in your base directory, then copy all the files and directories of Elgg's main folder to your server.


elgger@ftp.elgger.com - FileZilla
File Edit View Transfer Server Bookmarks Help New version available!
Quickconnect
Queued files
Failed transfers
Successful transfers

Host: ftp.elgger.com Username: elgger Password: •••••••• Port:

```
Response:     226-Options: -a -l
Response:     226 6 matches total
Status:       Directory listing successful
```

Local site: C:\Users\CJ\Desktop\elgg-1.8.0\

- Desktop
  - elgg-1.8.0
- Documents
- Downloads

Remote site: /

- /
  - backup
  - cgi
  - stats

Filename	Filesize	Filetype	Last modified
..			
_graphics		File folder	5/5/2011 7:55:21
actions		File folder	5/5/2011 7:55:13
documentation		File folder	5/5/2011 7:55:13
engine		File folder	5/5/2011 7:54:21
install		File folder	5/5/2011 7:55:14
js		File folder	5/5/2011 7:55:21
languages		File folder	5/5/2011 7:55:12
mod		File folder	5/5/2011 7:55:11

15 files and 11 directories. Total size: 1,096,329 bytes

Filename	Filesize	Filetype	Last modified	Perm
..				
backup		File folder	5/5/2011 8:00:2...	0705
cgi		File folder	5/5/2011 3:12:2...	0705
stats		File folder	5/5/2011 3:12:2...	0705
php.ini	1,100	Configurat...	5/5/2011 3:12:2...	0705

1 file and 3 directories. Total size: 1,100 bytes

Server/Local file	Direction	Remote file	Size	Priority	Status

Queue: empty

After you finish the transfer, check that there were not any failures (listed near the bottom of FileZilla). If there were, try transferring those files again. Another option is to FTP the ZIP file to the server and extract it there. The support staff for the server can tell you whether that is possible with your account.

Configuring the server

There are three steps to configuring the server:

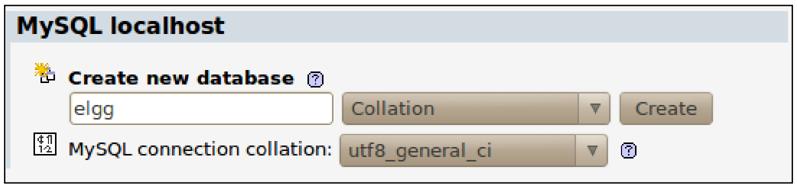
1. **Set the root and engine directory permissions**: The easiest way to run the installer is to give the web server permission to write to Elgg's root directory and the engine directory. Write down the current permissions on those directories and then set the permissions to 777. Doing this gives everyone with an account on the server (including the web server account) permission to write there. As soon as the installation is finished, reset the permissions to their original values. Don't forget!
2. **Create the data directory**: Elgg stores uploaded files like photos in this directory. For security reasons, it is best to put that directory outside of the server's web directory. If the web directory is `/var/www`, `/var/data/elgg`, then it is a good location for example. A good hosting provider will let you create a directory outside of the web directory. If you are using a shared hosting account, you may have to put the data directory in your web directory. See `http://docs.elgg.org/Data_directory` for instructions on this.
3. **Set the data directory permissions**: The web server needs to be able to create directories and write files to the data directory. On a Windows computer, it is unlikely that you need to change anything. With other operating systems, the best option is to make the web server the owner of the data directory. If you cannot do that, you will have to give everyone permission to write to it. You should be able to edit permissions using the FTP client.

Understanding permissions

The Elgg wiki has a tutorial on permissions with further instructions on setting the permissions of the data directory and changing the permissions for the installation. So if you want to understand why a permission of 777 gives everyone complete access to a file, visit `http://docs.elgg.org/Unix_Permissions_and_Elgg`.

Create the database

The last step before running the installer is creating a MySQL database for Elgg to store data. How you create the database depends on your server. If you are using your own computer and a package like XAMPP, use the phpMyAdmin tool to create the database.



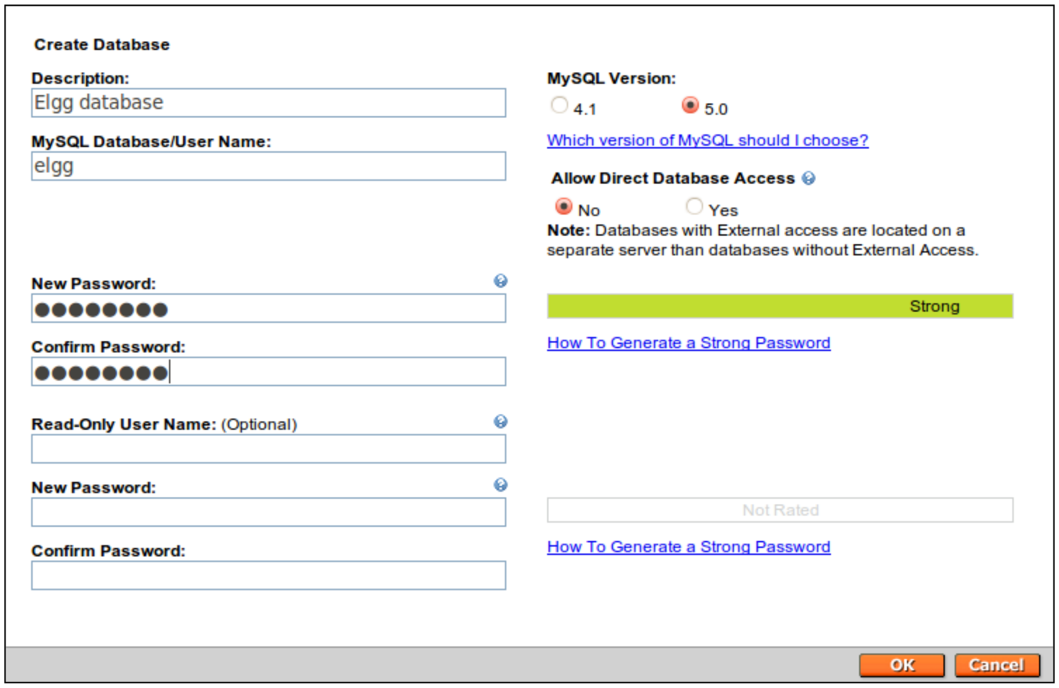
MySQL localhost

Create new database

elgg | Collation | Create

MySQL connection collation: utf8_general_ci

If you are using a hosting service, they will provide a tool for creating a database. The specific details vary per hosting service, but the general pattern is to log in to the **Control Panel**, find the button or link for creating a database, fill in the **Name** of the database, and click on the **Add** button. The tool may create a separate user account in addition to the database. Be sure to record all of this information for later use. The following is a screenshot from a popular hosting service's tool:



Create Database

Description: Elgg database

MySQL Database/User Name: elgg

New Password: ●●●●●●●●

Confirm Password: ●●●●●●●●

Read-Only User Name: (Optional)

New Password:

Confirm Password:

MySQL Version: 4.1 [✓ 5.0]

Which version of MySQL should I choose?

Allow Direct Database Access [✓ No] Yes

Note: Databases with External access are located on a separate server than databases without External Access.

Strong

How To Generate a Strong Password

Not Rated

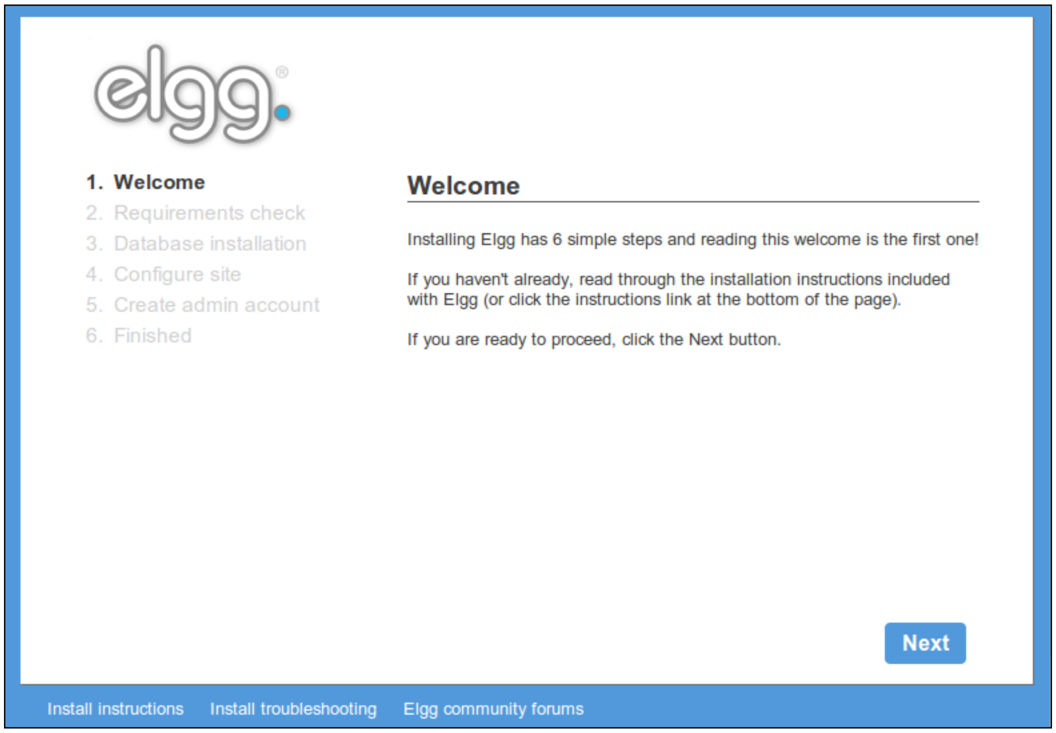
How To Generate a Strong Password

OK Cancel

Copyright © GoDaddy.com, Inc.

Run Elgg's installer

Once the server is configured and the database is created, you are ready to run the web-based installer. The installer creates tables in the Elgg database, sets basic parameters of the site like its name, and creates the administrative account. To launch the installer, use your web browser to view your site. The URL depends on the server and where you put Elgg's code. If you put the code in the root of your web directory, this will be your main URL. If you put the code in a subdirectory, you will need to add that to the URL. You should see the following:



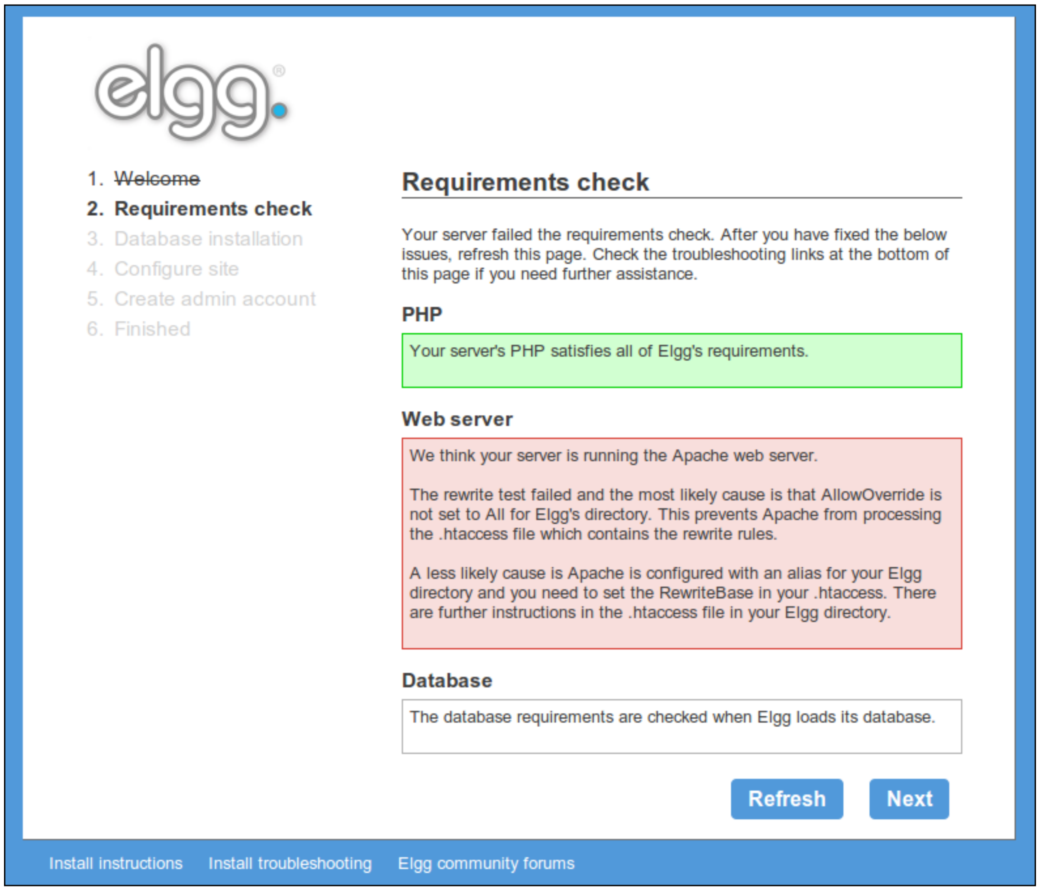
Welcome

Installing Elgg has 6 simple steps and reading this welcome is the first one!

If you haven't already, read through the installation instructions included with Elgg (or click the instructions link at the bottom of the page).

If you are ready to proceed, click the Next button.

There are six installation steps and each page will follow this same design with the steps listed on the left and helpful links at the bottom. After clicking on the **Next** button, the requirements checker runs and displays its results as shown in the following screenshot:



1. ~~Welcome~~
2. **Requirements check**
3. Database installation
4. Configure site
5. Create admin account
6. Finished

Requirements check

Your server failed the requirements check. After you have fixed the below issues, refresh this page. Check the troubleshooting links at the bottom of this page if you need further assistance.

PHP

Your server's PHP satisfies all of Elgg's requirements.

Web server

We think your server is running the Apache web server.

The rewrite test failed and the most likely cause is that AllowOverride is not set to All for Elgg's directory. This prevents Apache from processing the .htaccess file which contains the rewrite rules.

A less likely cause is Apache is configured with an alias for your Elgg directory and you need to set the RewriteBase in your .htaccess. There are further instructions in the .htaccess file in your Elgg directory.

Database

The database requirements are checked when Elgg loads its database.

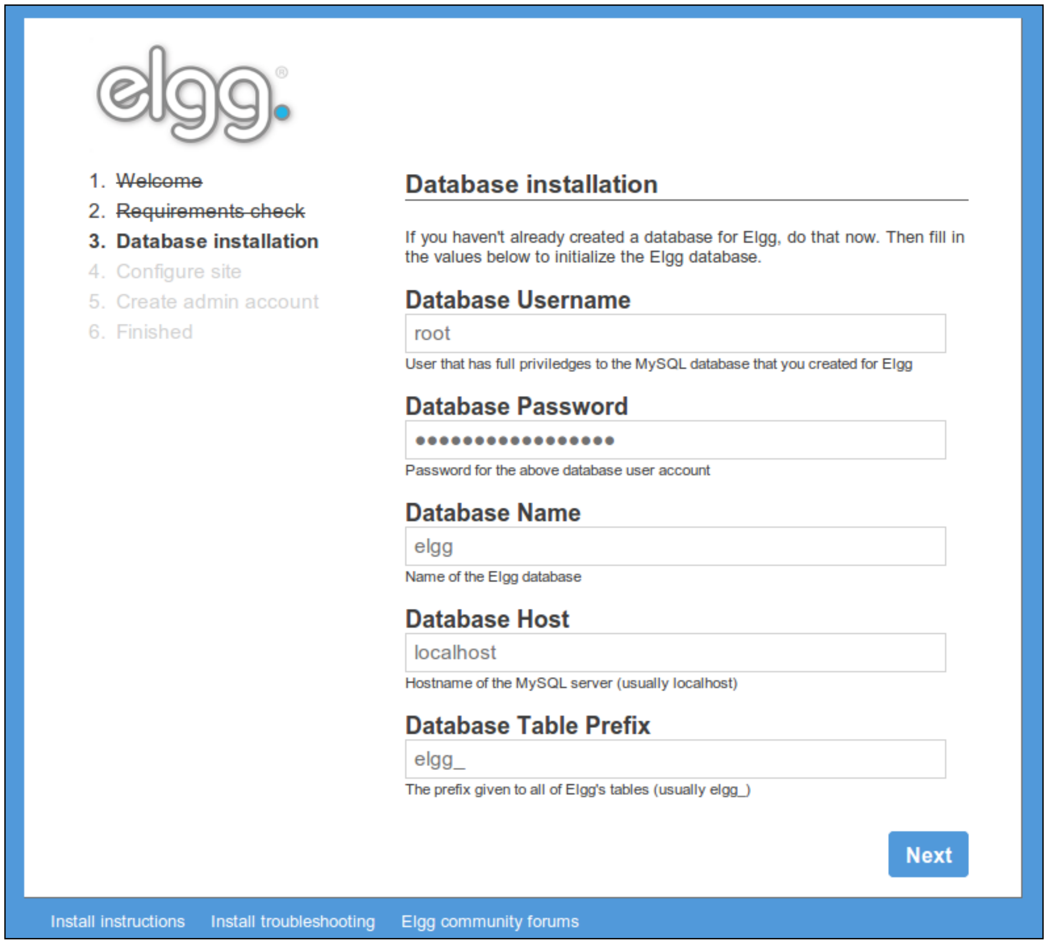
In the preceding screenshot, the check of the web server's URL rewriting has failed. This is the most common issue experienced by those installing Elgg and requires some minor server configuration to fix. Hints are given in the red error message with more detailed information available through the troubleshooting guide linked at the bottom of the page.

Install troubleshooting

There is extensive troubleshooting information on the Elgg wiki at `http://docs.elgg.org/Install_Troubleshooting`. This is the first place to seek help when experiencing a problem installing Elgg. To recheck the requirements after a configuration change, click on the **Refresh** button. If everything passes, you are ready to create the database tables.

Loading the database

This page asks for the information for the database that you created earlier. In most cases, you will not need to change the defaults for the database host and table prefix. Enter your settings and click on **Save**.



Database installation

If you haven't already created a database for Elgg, do that now. Then fill in the values below to initialize the Elgg database.

Database Username: root
User that has full priviledges to the MySQL database that you created for Elgg

Database Password: ●●●●●●●●●●●●●●●●●
Password for the above database user account

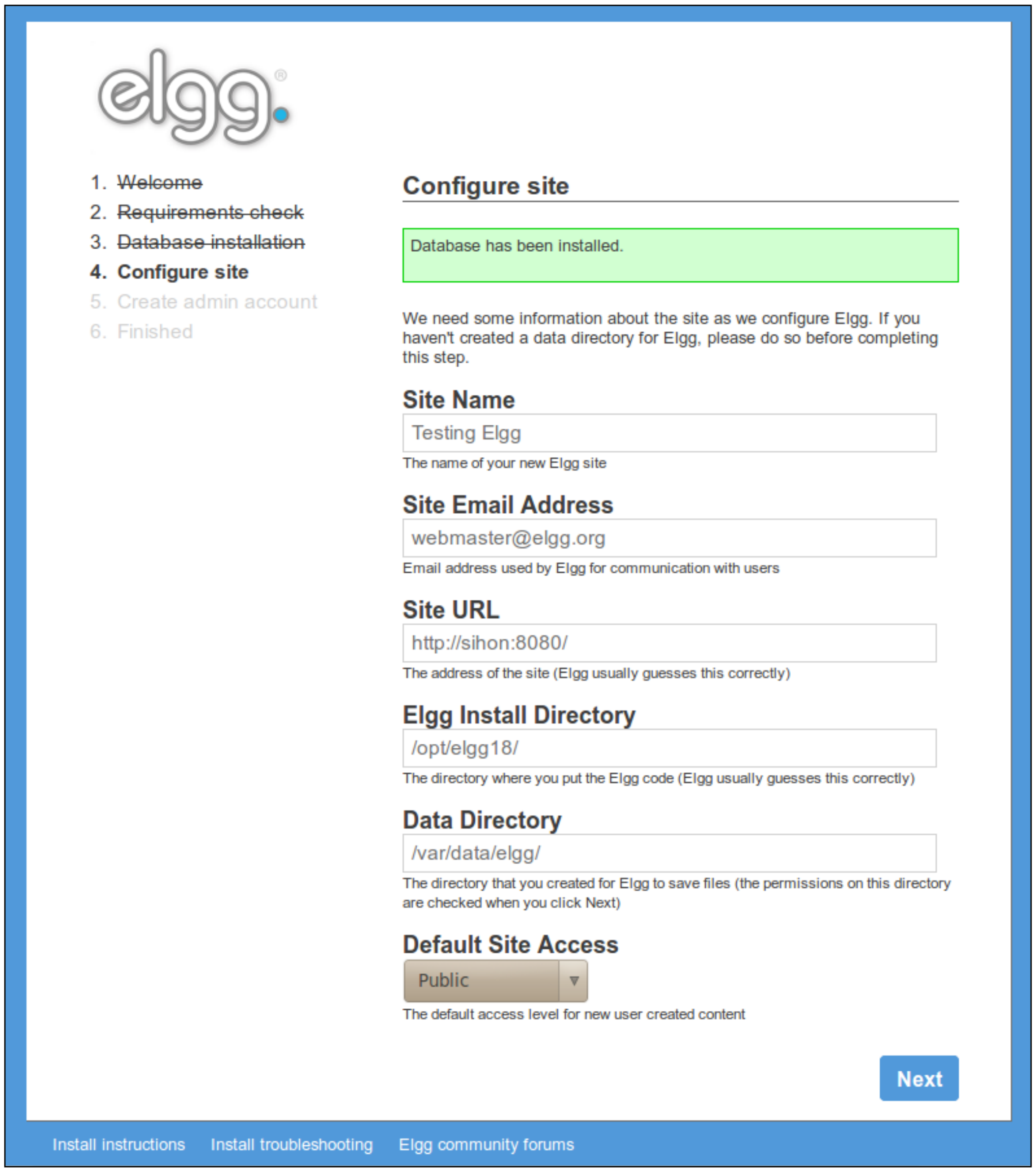
Database Name: elgg
Name of the Elgg database

Database Host: localhost
Hostname of the MySQL server (usually localhost)

Database Table Prefix: elgg_
The prefix given to all of Elgg's tables (usually elgg_)

Configure Elgg

The next page asks for settings about your site. You should only have to set the name of the site, the site e-mail address, and the data directory. Elgg is able to detect the rest of the settings. The e-mail address is not your own, but the e-mail address that you want notification e-mails to be sent from.


1. ~~Welcome~~
2. ~~Requirements check~~
3. ~~Database installation~~
4. **Configure site**
5. Create admin account
6. Finished

Configure site

Database has been installed.

We need some information about the site as we configure Elgg. If you haven't created a data directory for Elgg, please do so before completing this step.

Site Name
Testing Elgg
The name of your new Elgg site

Site Email Address
webmaster@elgg.org
Email address used by Elgg for communication with users

Site URL
http://sihon:8080/
The address of the site (Elgg usually guesses this correctly)

Elgg Install Directory
/opt/elgg18/
The directory where you put the Elgg code (Elgg usually guesses this correctly)

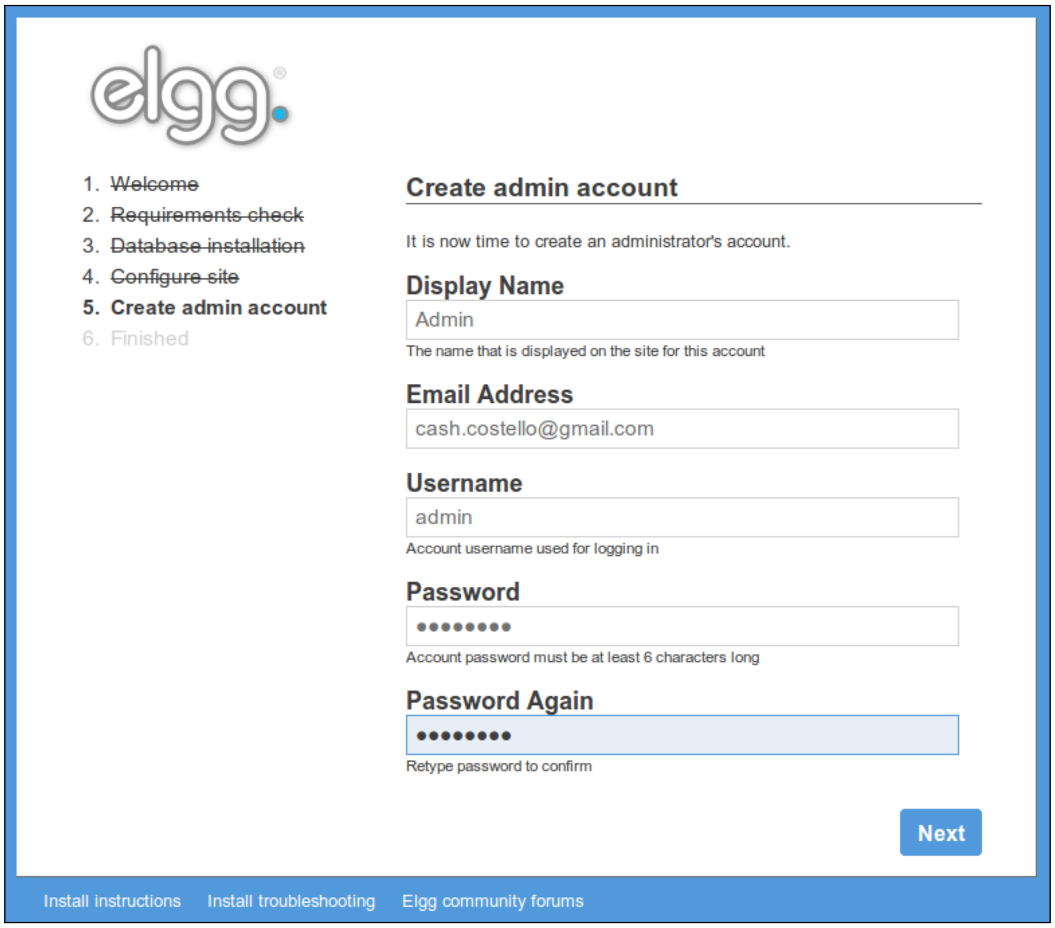
Data Directory
/var/data/elgg/
The directory that you created for Elgg to save files (the permissions on this directory are checked when you click Next)

Default Site Access
Public
The default access level for new user created content

These settings are not final. You can change them at any time after the installation in the administration area of Elgg. We will discuss the administration area in more detail in the next chapter.

Creating the admin account

The second to the last step of the installer is creating the administrative account. Be sure to record your username and password. If your server does not have e-mail set up, it is very difficult to reset a forgotten admin account password. After you have entered the information, click on **Next**. This takes you to the last step, which is just a message letting you know the site is ready to be used.



1. ~~Welcome~~
2. ~~Requirements check~~
3. ~~Database installation~~
4. ~~Configure site~~
5. **Create admin account**
6. Finished

Create admin account

It is now time to create an administrator's account.

Display Name
Admin
The name that is displayed on the site for this account

Email Address
cash.costello@gmail.com

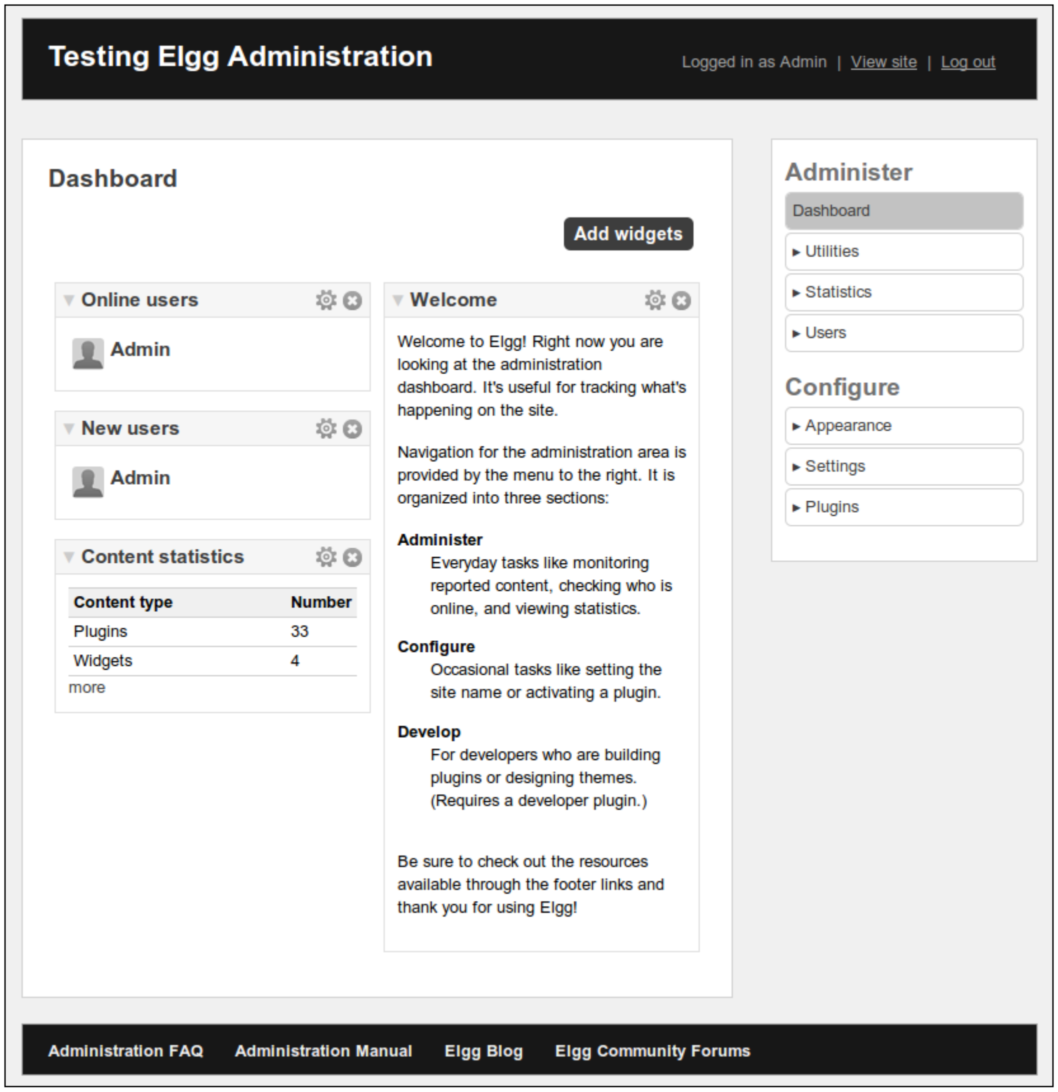
Username
admin
Account username used for logging in

Password
••••••••
Account password must be at least 6 characters long

Password Again
••••••••
Retype password to confirm

All done!

With the administrative account created, you are now ready to use the site. All you have to do is click on **Next** and Elgg logs you into the site. The first page it takes you to is the dashboard of the administration area. We will be taking a tour of this area in the next chapter. If you want to see what the user-facing portion of the site looks like, click on the **View site** link in the header.



Content type	Number
Plugins	33
Widgets	4

Welcome to Elgg! Right now you are looking at the administration dashboard. It's useful for tracking what's happening on the site.

Navigation for the administration area is provided by the menu to the right. It is organized into three sections:

Administer
Everyday tasks like monitoring reported content, checking who is online, and viewing statistics.

Configure
Occasional tasks like setting the site name or activating a plugin.

Develop
For developers who are building plugins or designing themes. (Requires a developer plugin.)

Be sure to check out the resources available through the footer links and thank you for using Elgg!

Summary

This chapter guided you through the installation of Elgg. With the site set up, you are ready to start using it. The next three chapters help you explore what Elgg can do and provide commentary that assists you in designing and building your own Elgg site.

3
A Tour of Your First Elgg Site

Now that you have installed Elgg, it is time to get some hands-on experience. This chapter takes you on a guided tour of Elgg, providing you with a broad overview of what Elgg does. We describe how to create test users, fill out their profiles, and find them some friends. The content sharing and communication tools are covered next. We round out the tour with stops in user settings and administration.

Treat your site as a test site. You can create users, fill out profiles, try various site settings, and delete users knowing that you can wipe the site and create another in a few minutes.

While working through the tour, take notes on what you would like to change or add. Those notes will be extremely useful when designing your site. This and future chapters include design ideas and hints to help you start planning. You will find that some customizations are easy to do (maybe by installing a plugin available in Elgg's community plugin repository). Other modifications will require development skills and knowledge of Elgg's plugin system.

Topics covered in this chapter include:

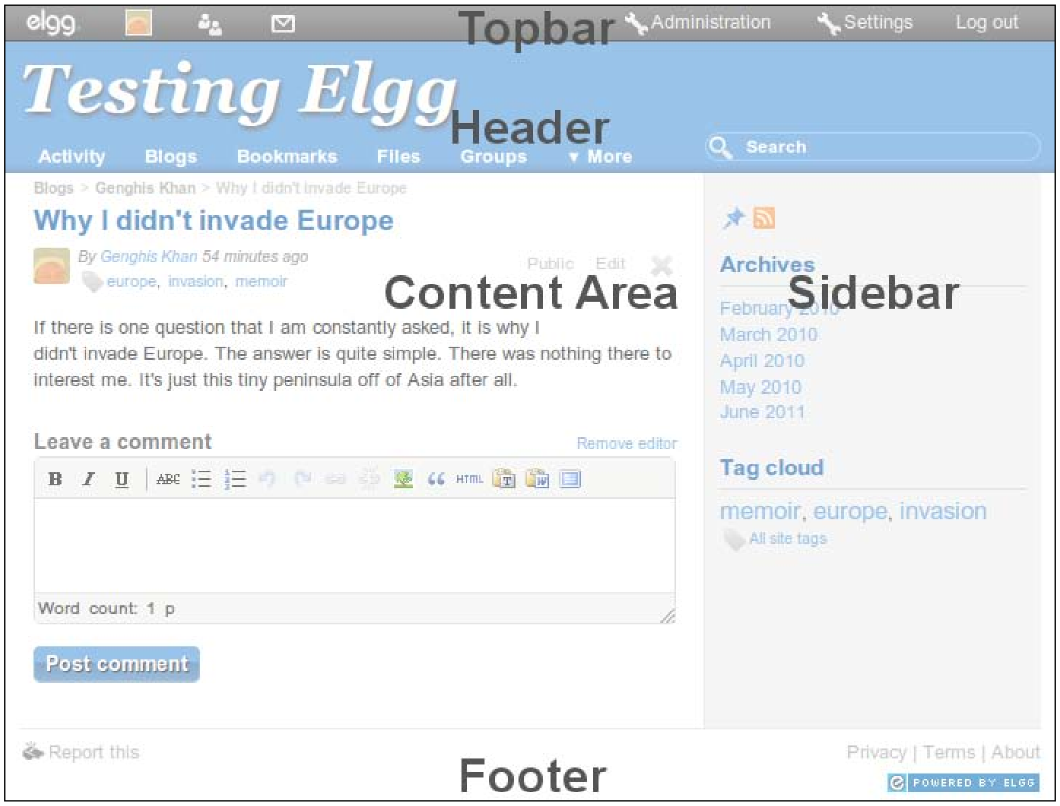
- User registration and profile creation
- Activity streams
- Sharing content through blogs, bookmarks, files, and microblogging
- Configuring user settings
- Administering an Elgg site

This chapter is an orientation to Elgg and so does not spend a lot of time on any one part. The next two chapters cover each of the contenting sharing plugins mentioned here in much more detail. Then *Chapter 6, Finding and Using Plugins* explains how to find and install plugins so be sure to read that before you begin downloading new plugins.

Getting around

After you installed Elgg, you probably took a look around and noticed that there is a user area and a separate administration area. Each area has its own theme. The link to switch between the two areas is at the top of the page and is only available to administrators.

The web pages have a consistent layout throughout the site. The following is a screenshot of a sample web page with each component labeled. These component labels are used throughout this book.



If there is one question that I am constantly asked, it is why I didn't invade Europe. The answer is quite simple. There was nothing there to interest me. It's just this tiny peninsula off of Asia after all.

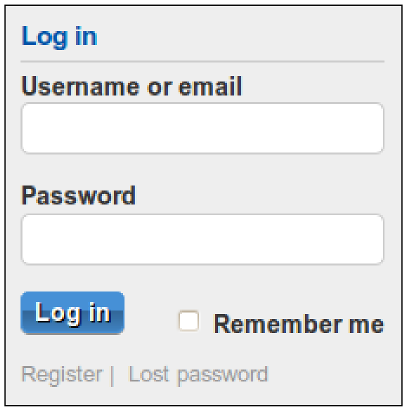
- **Topbar**: The aptly named topbar is located at the top of the page and is used for navigation.
- **Header**: The site name or logo is located below the topbar in the header. A site menu for navigating to Elgg's different tools is also in the header.
- **Content Area**: Most pages have two columns with the content in the wider column.

- **Sidebar**: The narrow column on the right has a menu specific to a particular area of the site. For example, on a blog page the sidebar menu has links for viewing and interacting with blogs.
- **Footer**: At the bottom of the page is the footer. It often contains another site navigation menu or links to site specific pages such as a contact form or a privacy statement.

This is the default layout of Elgg and can be customized as detailed in *Chapter 9*. If your Elgg site does not look exactly like the screenshots in this book, do not be alarmed. The developers of Elgg tend to redesign the default theme of Elgg after every two or three major releases, but the underlying functionality should be very similar.

Registering users

Before we can test Elgg's social networking capability we need to add test users to our site. The link to the registration page is found on the **Log in** box on the front page of the site (which is only displayed when you are logged out).



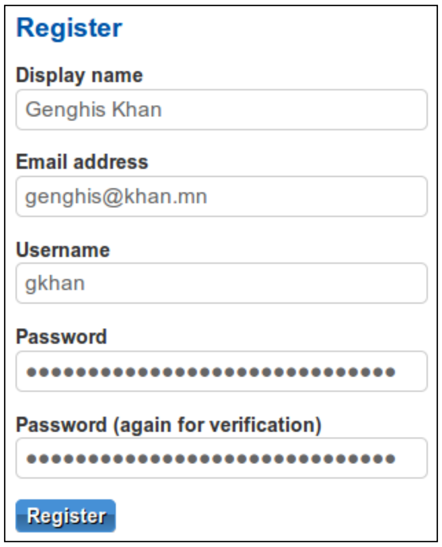
The default registration process for Elgg consists of three steps:

1. Fill out the registration form.
2. Receive an e-mail with a validation link.
3. Click the validation link to confirm the e-mail address belongs to you.

The validation process assumes that your server is properly configured for sending e-mail. If this is not the case, users can be manually created in the administration area. This is also the quickest method for creating users because it skips the validation step. In order to manually create a user, switch to the administration area and find the **Add New User** link on the sidebar under **Administer | Users**.

Multiple test users

The best way to test the social functionality of Elgg is by controlling multiple test users at a time. This is possible if you have more than one web browser on your computer. One user could be logged in using Firefox, another with Chrome, and so on until you run out of web browsers. Testing will proceed more quickly as you will not need to constantly log in and out of different user accounts. It is also important to test with users who are not administrators so that you see what your users will see.


Register

Display name: Genghis Khan

Email address: genghis@khan.mn

Username: gkhan

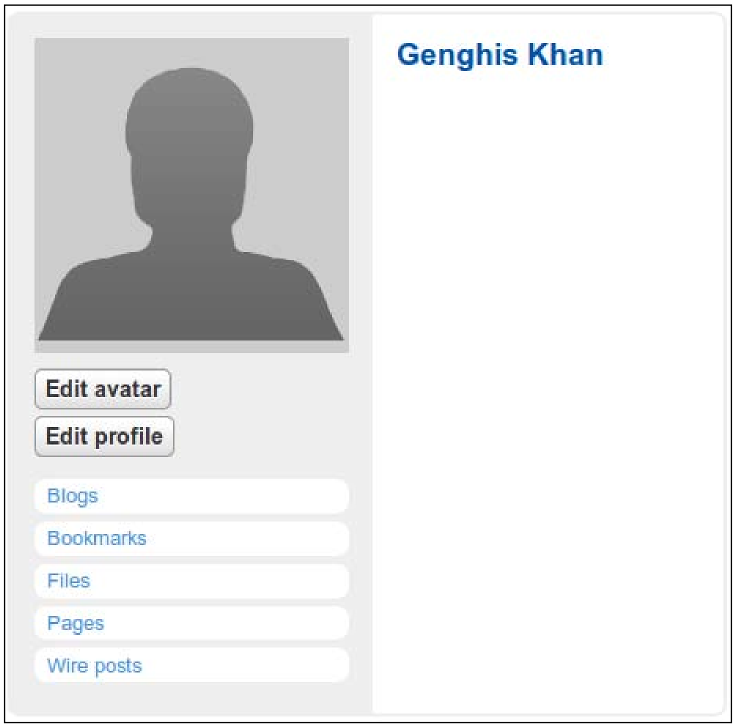
Password: ••••••••••••••••••••••••••••••

Password (again for verification): ••••••••••••••••••••••••••••••

Later in the chapter, we will explore the administration area. There you can turn off the user validation plugin or turn off registration completely.

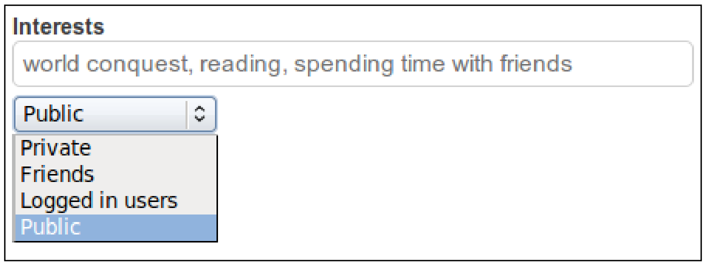
User profiles and avatars

Each user has a profile page that includes an avatar, profile information, and widgets. A user's profile is reached by clicking on the user's avatar anywhere in the site. Users can navigate to their own profile by clicking on the mini avatar in the topbar next to the Elgg logo.



Profile information

To add some information to that very empty profile, click on the **Edit profile** button. Elgg comes with a default set of profile fields already defined. Some fields are tags like **Interests** and **Skills**. Clicking on one of these tags leads to a web page that lists users who share that interest or skill. Other fields are links to web pages or plain text boxes.



Interests

world conquest, reading, spending time with friends

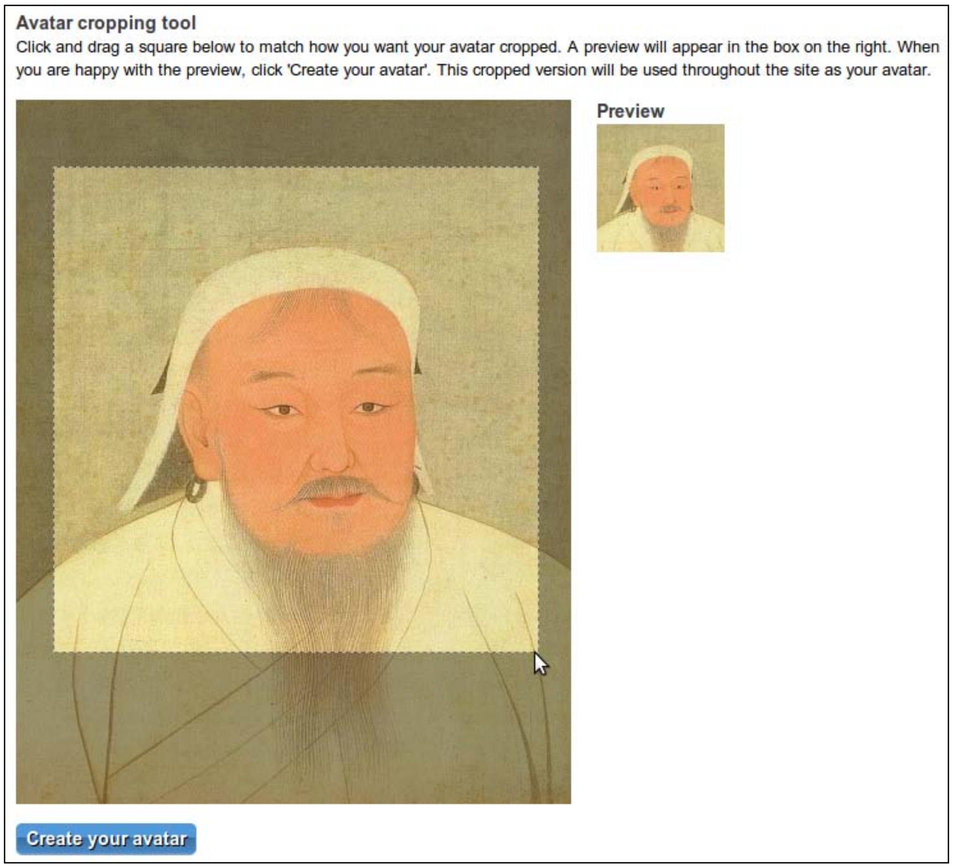
Public

Profile field permissions

Did you notice that each profile field has a drop-down menu underneath it? Elgg has a robust permissions system so that users can control who has access to content. Whether it's a blog post, uploaded file, or a profile field, the creator of the content gets to decide who can see it.

Avatar

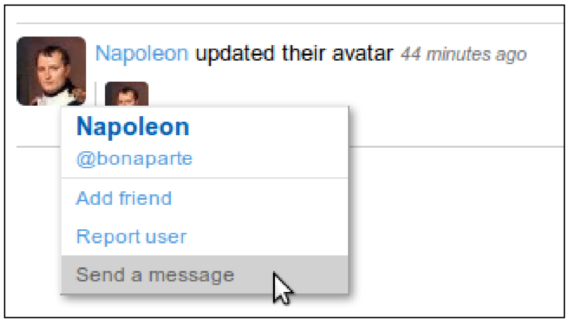
Users are represented throughout the site by an avatar (also called a profile icon). The avatar appears next to comments, forum posts, or any other content posted by a user. A user sets the avatar by clicking on the **Edit Avatar** link underneath the avatar on the profile page. The user selects an image from their computer and uploads it to the server. Then Elgg's handy cropping tool is used to select a particular region for the avatar.



Avatar cropping tool

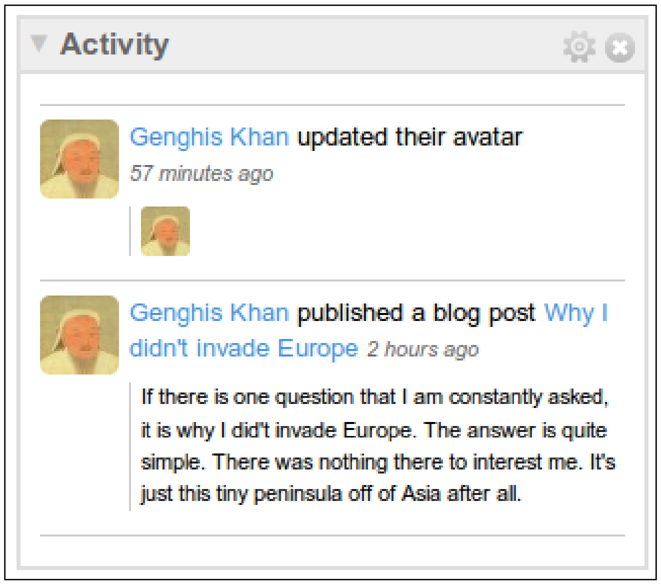
Click and drag a square below to match how you want your avatar cropped. A preview will appear in the box on the right. When you are happy with the preview, click 'Create your avatar'. This cropped version will be used throughout the site as your avatar.

An important feature that many people new to Elgg miss is that the avatar has its own drop-down menu. This menu includes commonly used links. To try it, move your mouse cursor over a user's avatar and click on the small arrow icon that appears over the avatar.


Napoleon updated their avatar 44 minutes ago

Profile widgets

Users can not only set their profile information and avatar, but they can also configure widgets for their profile. Widgets are small web elements that display all kinds of content: a list of friends, blog posts, group membership, YouTube videos, MP3 players, and more. The following is an activity widget that lets viewers know what a user has been doing on the site:

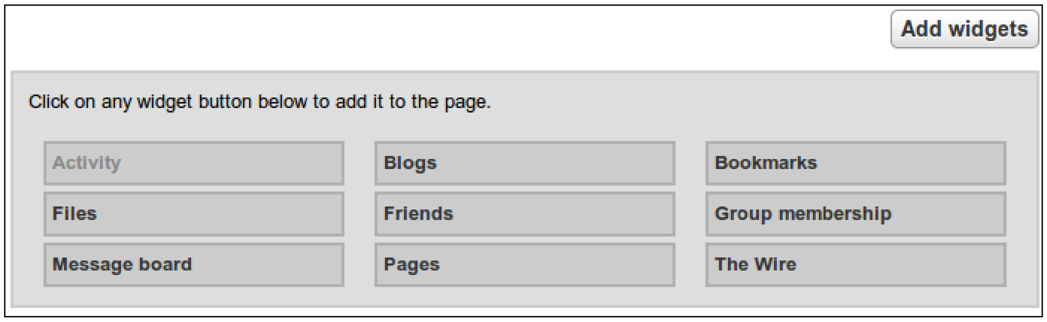


Genghis Khan updated their avatar
57 minutes ago

Genghis Khan published a blog post Why I didn't invade Europe 2 hours ago

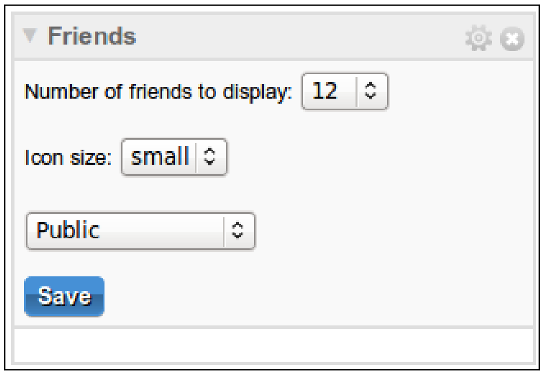
If there is one question that I am constantly asked, it is why I did't invade Europe. The answer is quite simple. There was nothing there to interest me. It's just this tiny peninsula off of Asia after all.

In order to add a widget, click on the **Add widgets** button on the top right of the profile page. Next, click any of the widget buttons to add it. Once the widget has been added, it can be moved to any of the three columns by dragging it by its title. Clicking on the **Add widgets** button again hides the widget selection panel.



Click on any widget button below to add it to the page.

Each widget has its own settings that can be accessed by clicking on the edit link on the title bar of the widget. For example, the **Friends** widget has settings for the number of displayed friends, the size of the icons, and who can view the widget.

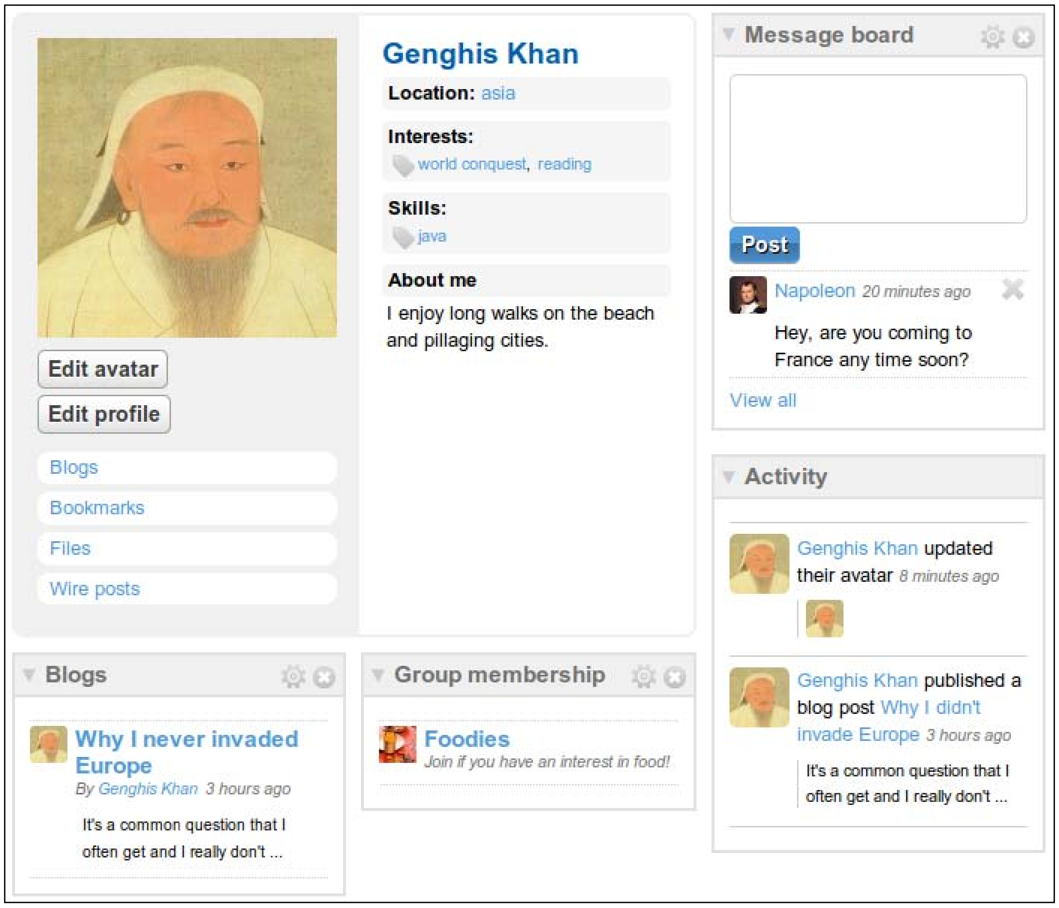


Number of friends to display: 12

Icon size: small

Public

When you are done adding and configuring the widgets, the empty look of the profile page will have been replaced with interesting information about the user.



Genghis Khan

Location: asia

Interests: world conquest, reading

Skills: java

About me

I enjoy long walks on the beach and pillaging cities.

Napoleon 20 minutes ago

Hey, are you coming to France any time soon?

Genghis Khan updated their avatar 8 minutes ago

Genghis Khan published a blog post Why I didn't invade Europe 3 hours ago

It's a common question that I often get and I really don't ...

Why I never invaded Europe

By Genghis Khan 3 hours ago

It's a common question that I often get and I really don't ...

Foodies

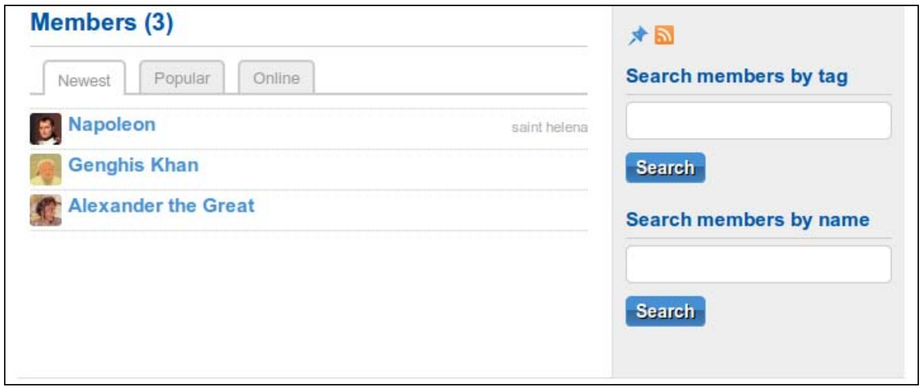
Join if you have an interest in food!

Friends

As Elgg is a platform for building social networking sites, forming relationships is an essential capability. These relationships are called "friends" on Elgg, similar to Facebook, but they work more like "followers" on Twitter. It is a one-way relationship and does not require confirmation by the person being "friended".

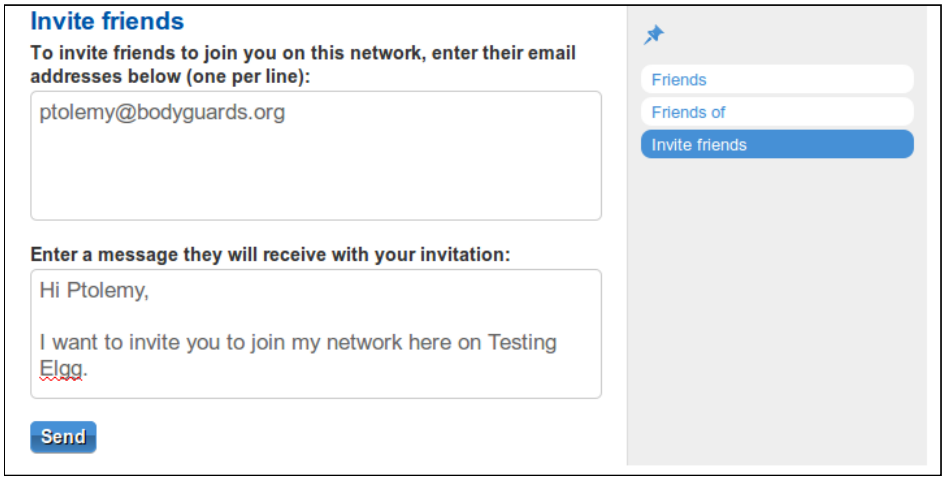
A user can add a friend by visiting a person's profile and clicking on the **Add friend** link. The drop-down menu on a user's avatars can also be used to add friends. The user who is being "friended" is notified by an e-mail. Removing a friend works the same way except that the user is not notified.

A good place to look for other users is on the **Members** page. Navigate to this page by using the **More** tab in the site menu.



The members list can be sorted, by when people joined the site, who has the most friends, and who is currently online. The members page also has two search boxes: one for searching by profile field tags such as interests and location and the other for searching by name.

Users can also invite people they know to join the site from Elgg. This is a great way to grow a site since your users are recruiting new members for you. The **Invite friends** link is on the sidebar of the **Friends** page, which is reached by clicking on the friends icon in the topbar.


Invite friends

To invite friends to join you on this network, enter their email addresses below (one per line):

ptolemy@bodyguards.org

Enter a message they will receive with your invitation:

Hi Ptolemy,

I want to invite you to join my network here on Testing Elgg.

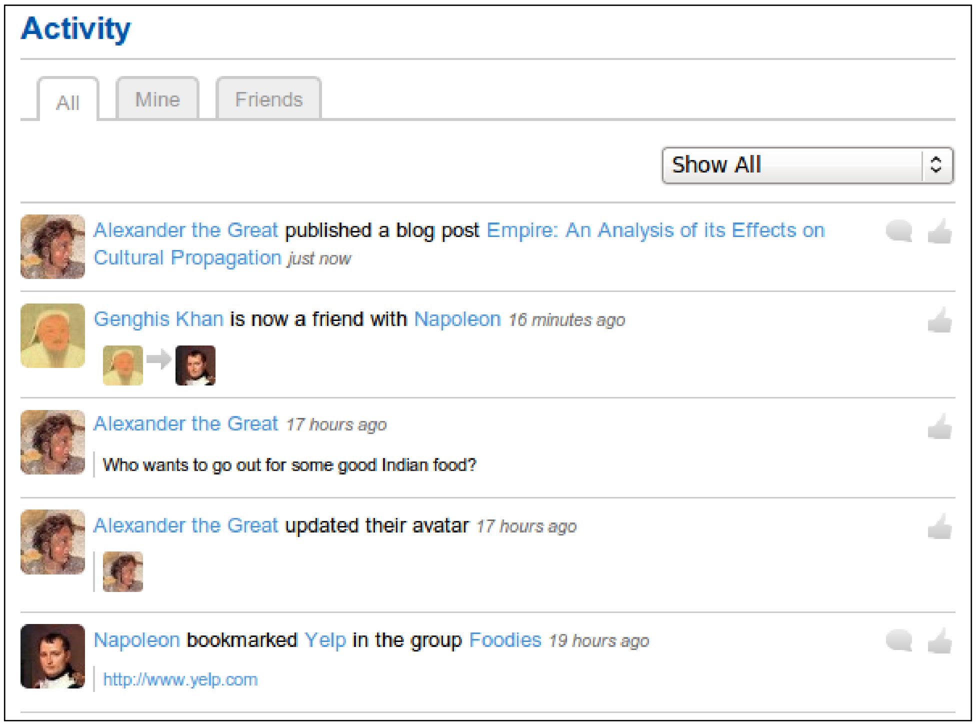
Invited people receive an e-mail with a personalized message and a link to join the site. When an invited person joins, the two users are automatically made friends.

Activity stream

A great way to see everything that is happening on a site is Elgg's activity stream. The stream includes short snippets of information on what the members of the site are doing. The page has tabs so that users can choose between seeing everyone's activity, just their friends' activity or their own. They can also filter by content type (for example, only blog posts). The activity stream respects the permission settings on content so that if a user sets a blog post to be for friends only, that blog post only appears in the activity stream of those friends.

These types of pages are very important for social sites because they make the site appear alive with activity, and they can help people find interesting content that they might have missed otherwise. For many users, this is the first page that they check after they log in.

In additional to viewing content, users can also interact with the content in the activity stream. On the right of many items is a thumbs up icon for "liking" the content. If an item has been liked by at least one person, a count is displayed. Clicking on the count displays a list of users who have liked the item. Another kind of interaction supported is responding to the content on the activity page. The speech bubble is a link for adding a comment or replying to a forum post.



Alexander the Great published a blog post Empire: An Analysis of its Effects on Cultural Propagation *just now*

Genghis Khan is now a friend with Napoleon *16 minutes ago*

Alexander the Great *17 hours ago*

Who wants to go out for some good Indian food?

Alexander the Great updated their avatar *17 hours ago*

Napoleon bookmarked Yelp in the group Foodies *19 hours ago*

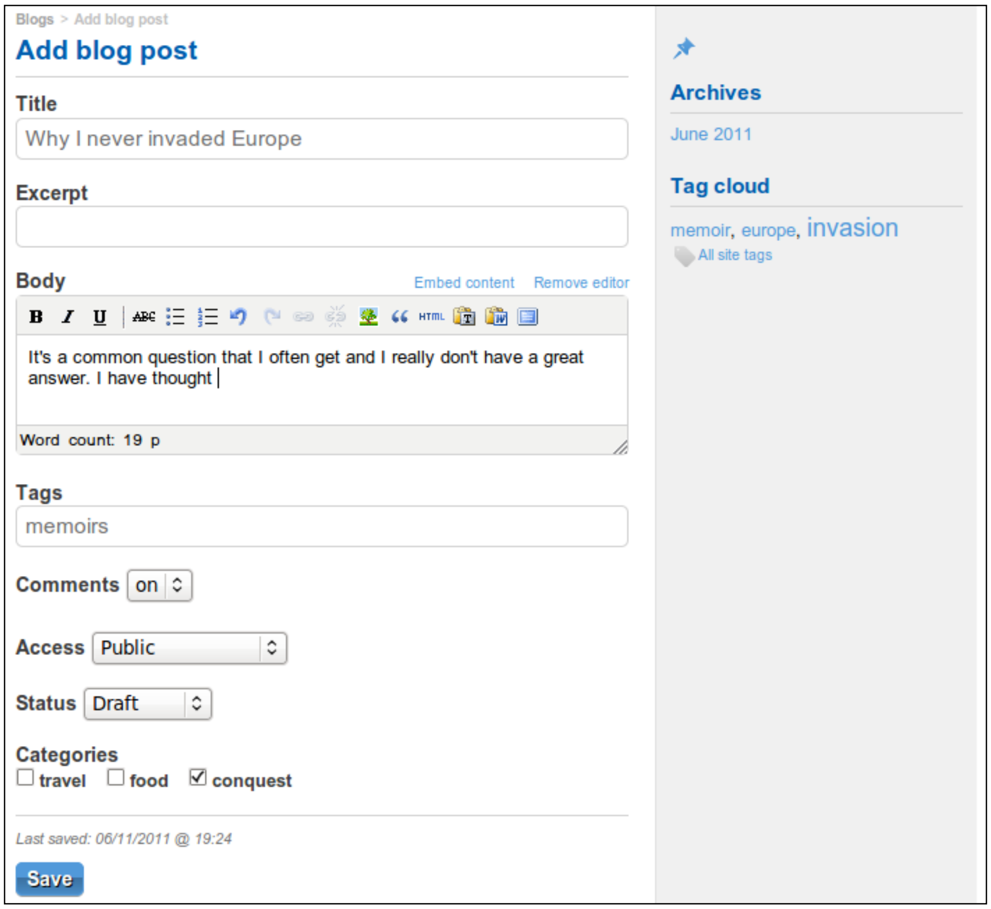
http://www.yelp.com

Tools

Elgg includes a set of tools for sharing content, collaborating, and communicating. Each tool is provided by a plugin. This section gives a quick overview of these tools while *Chapter 4, Sharing Content* and *Chapter 5, Communities, Collaboration, and Conversation* cover the tools in greater detail.

Blogs

Elgg has a basic blogging tool that is very easy to use. Users can write blog posts and comment on them. The tool supports saving drafts, embedding photos, adding tags, and optional commenting. An editor for marking text as bold or creating lists is available. A user can also select from categories defined by the Site Administrator. When a user publishes a blog post, it appears in the activity stream and notifications are sent out.


Blogs > Add blog post

Add blog post

Title: Why I never invaded Europe

Excerpt:

Body

It's a common question that I often get and I really don't have a great answer. I have thought

Word count: 19 p

Tags: memoirs

Comments: on

Access: Public

Status: Draft

Categories: [ ] travel [ ] food [✓] conquest

Last saved: 06/11/2011 @ 19:24

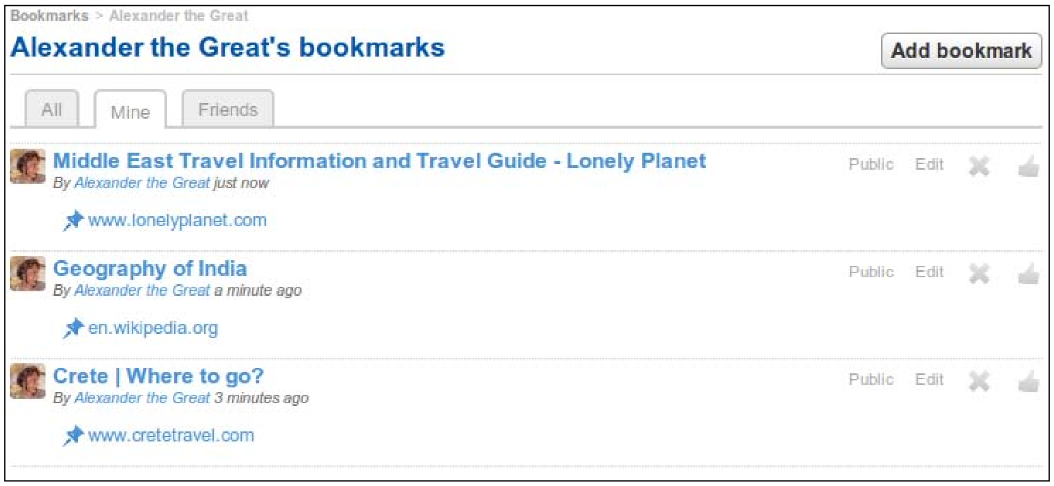
Archives: June 2011

Tag cloud: memoir, europe, invasion

We have already mentioned that much of the functionality that is visible to users is provided by plugins. This description of the blogging tool includes features supplied by the following plugins: blogs, categories, embed, and TinyMCE editor. The last three plugins integrate with most of the tools in this section.

Bookmarks

Social bookmarking enables users to share links to websites with each other. Delicious, Digg, and Reddit are examples of social bookmarking sites on the Web. Just as with the other content sharing tools, commenting, tagging, and notifications are supported.


Bookmarks > Alexander the Great
Alexander the Great's bookmarks
Add bookmark
All
Mine
Friends
Public
Edit

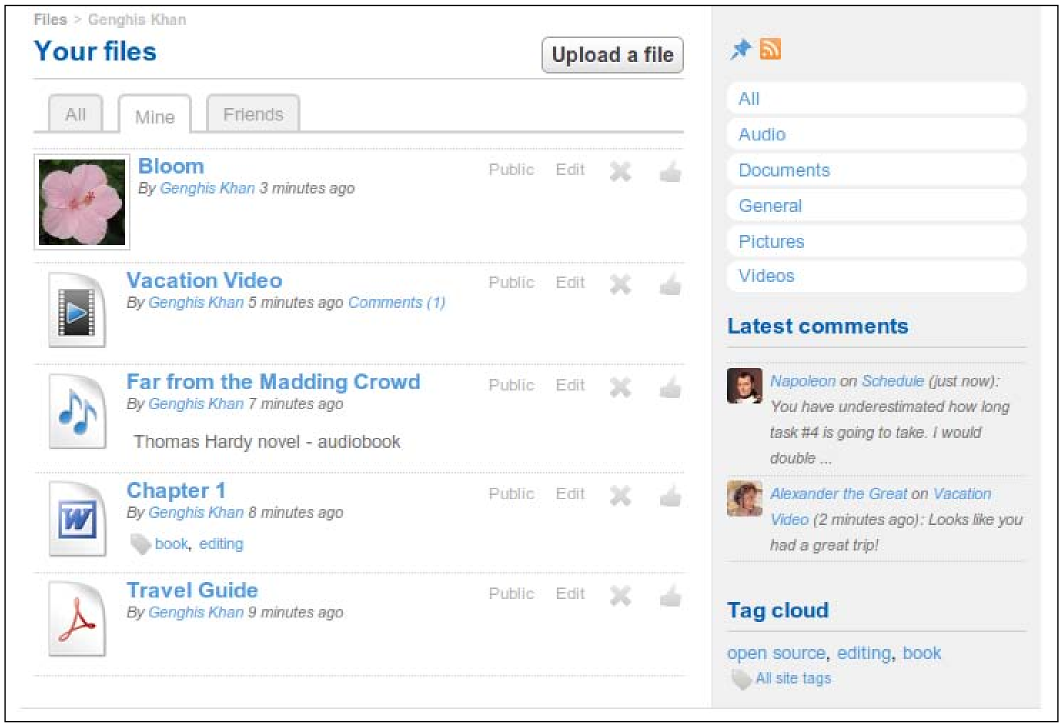
Middle East Travel Information and Travel Guide - Lonely Planet
By Alexander the Great just now
www.lonelyplanet.com

Geography of India
By Alexander the Great a minute ago
en.wikipedia.org

Crete | Where to go?
By Alexander the Great 3 minutes ago
www.cretetravel.com

Files

Users can upload and share any kind of file with this tool: Word documents, Excel spreadsheets, photos, audio files, and videos. The icons assigned to each file correspond to its type and the tool also creates a categorized list of files by type in the sidebar.



Files > Genghis Khan

Your files

Bloom
By Genghis Khan 3 minutes ago
Public Edit

Vacation Video
By Genghis Khan 5 minutes ago Comments (1)
Public Edit

Far from the Madding Crowd
By Genghis Khan 7 minutes ago
Thomas Hardy novel - audiobook
Public Edit

Chapter 1
By Genghis Khan 8 minutes ago
book, editing
Public Edit

Travel Guide
By Genghis Khan 9 minutes ago
Public Edit

Latest comments

Napoleon on Schedule (just now): You have underestimated how long task #4 is going to take. I would double ...

Alexander the Great on Vacation Video (2 minutes ago): Looks like you had a great trip!

Tag cloud

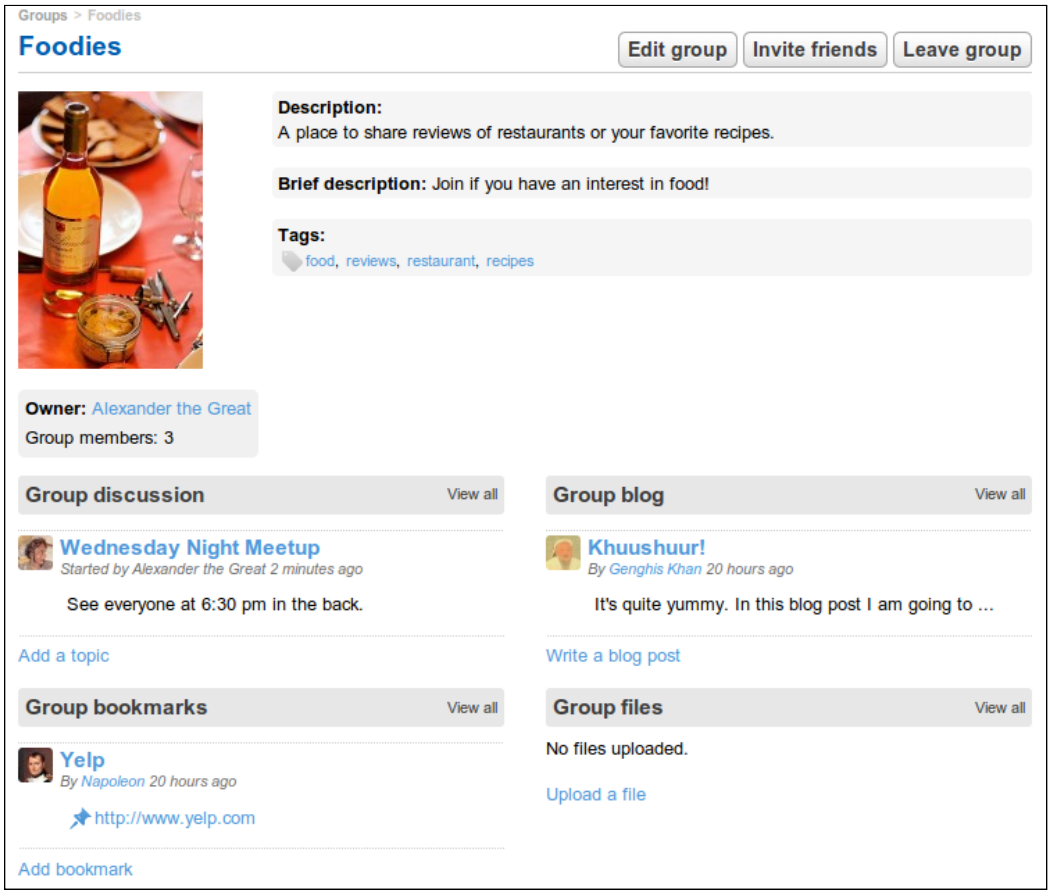
open source, editing, book
All site tags

Groups

The groups tool is the most popular tool in Elgg as it makes it easy for users to form virtual communities. The communities often organize around a shared interest, background, job, or purpose. As an example, on a social networking site for sports fans, people might form groups focused on their favorite team.

A group gets a set of tools to encourage collaboration and communication: blog, bookmarking, file sharing, forum, and pages. Other tools can be added through plugins. The group profile page, shown in the following screenshot, provides a quick overview of what the group is about and what has been happening in it.

The owner of the group can invite members of the site to join the group. The owner can also set the group to be closed, which means that members must be invited in order to join. As with content created by individuals, group content shows up in activity streams and generates notifications for group members.



Description:
A place to share reviews of restaurants or your favorite recipes.

Brief description: Join if you have an interest in food!

Tags:
food, reviews, restaurant, recipes

Owner: Alexander the Great
Group members: 3

Wednesday Night Meetup
Started by Alexander the Great 2 minutes ago

See everyone at 6:30 pm in the back.

Khuushuur!
By Genghis Khan 20 hours ago

It's quite yummy. In this blog post I am going to ...

Yelp
By Napoleon 20 hours ago

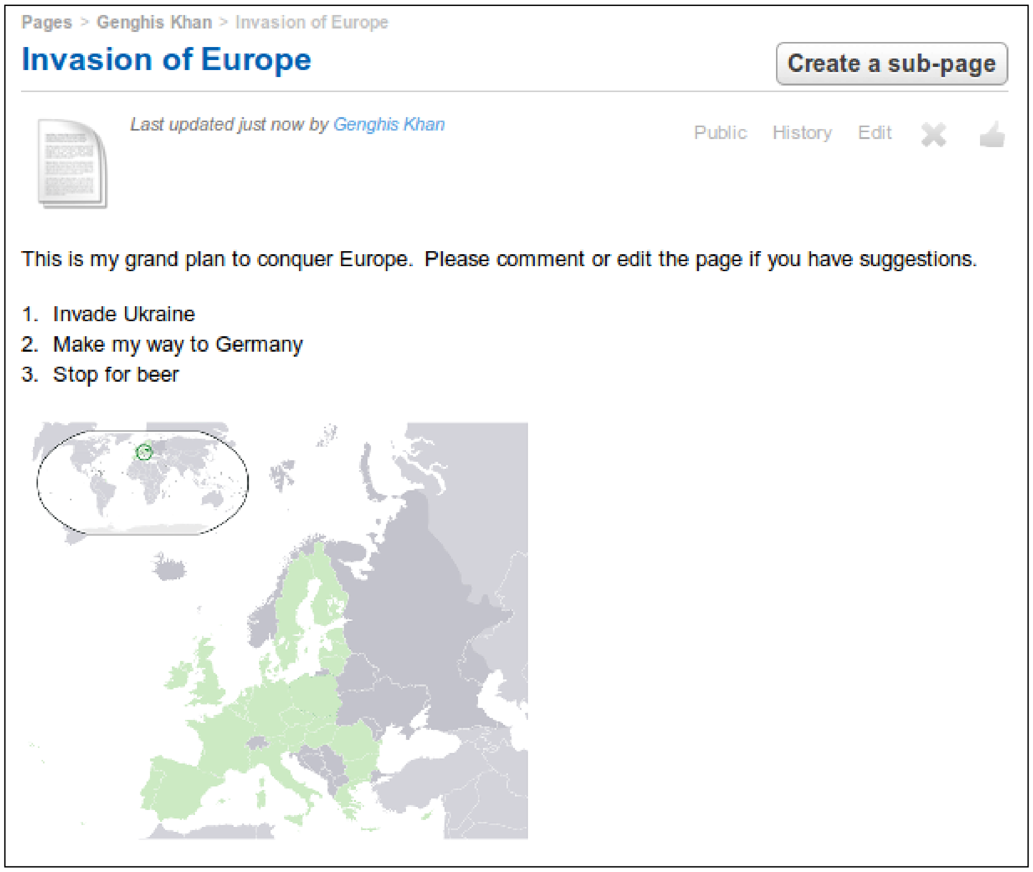
http://www.yelp.com

No files uploaded.

Pages

The pages tool lets users collaboratively edit a web page of content. It works a lot like a wiki without requiring the use of wiki markup (which many people have trouble using). Instead of wiki markup, the same editor that was used for blogs is available for editing the pages.

The creator of a page decides who has permission to edit the page and who can view it. Every time a page is saved a revision is created so that users can look at the history of changes.



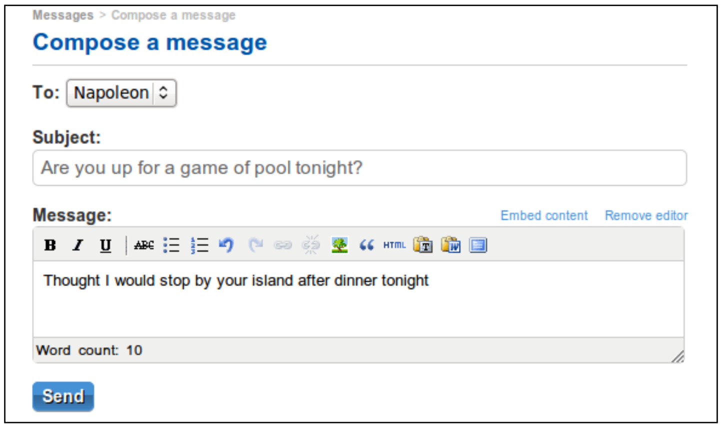
Last updated just now by Genghis Khan

This is my grand plan to conquer Europe. Please comment or edit the page if you have suggestions.

1. Invade Ukraine
2. Make my way to Germany
3. Stop for beer

Messages

A private messaging capability is provided by the messages tool. With it, users can send messages to their friends on the site. It works just like e-mail without requiring the users to know each other's e-mail address.



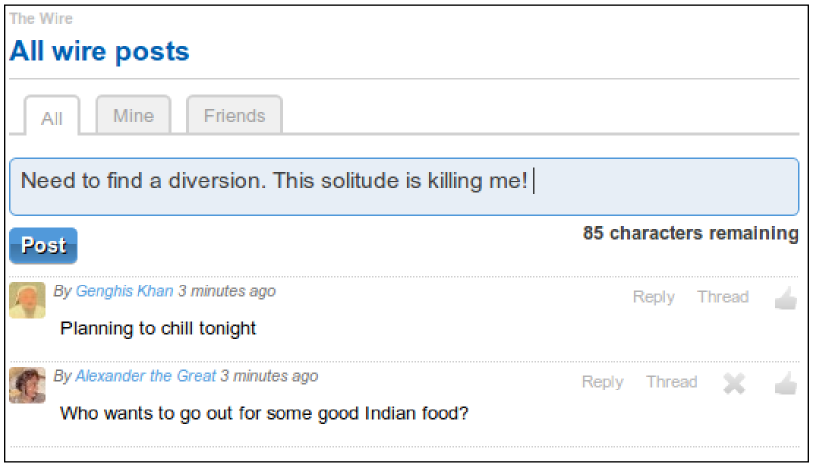
Subject: Are you up for a game of pool tonight?

Message: Thought I would stop by your island after dinner tonight

Word count: 10

The wire

The wire is a microblogging tool similar to Twitter. Users can post messages of up to 140 characters. Using the Twitter API plugin, distributed with Elgg, it is possible for users to have their wire posts also pushed to their Twitter accounts. *Chapter 5* provides instructions for configuring this.



Need to find a diversion. This solitude is killing me!

85 characters remaining

By Genghis Khan 3 minutes ago

Planning to chill tonight

By Alexander the Great 3 minutes ago

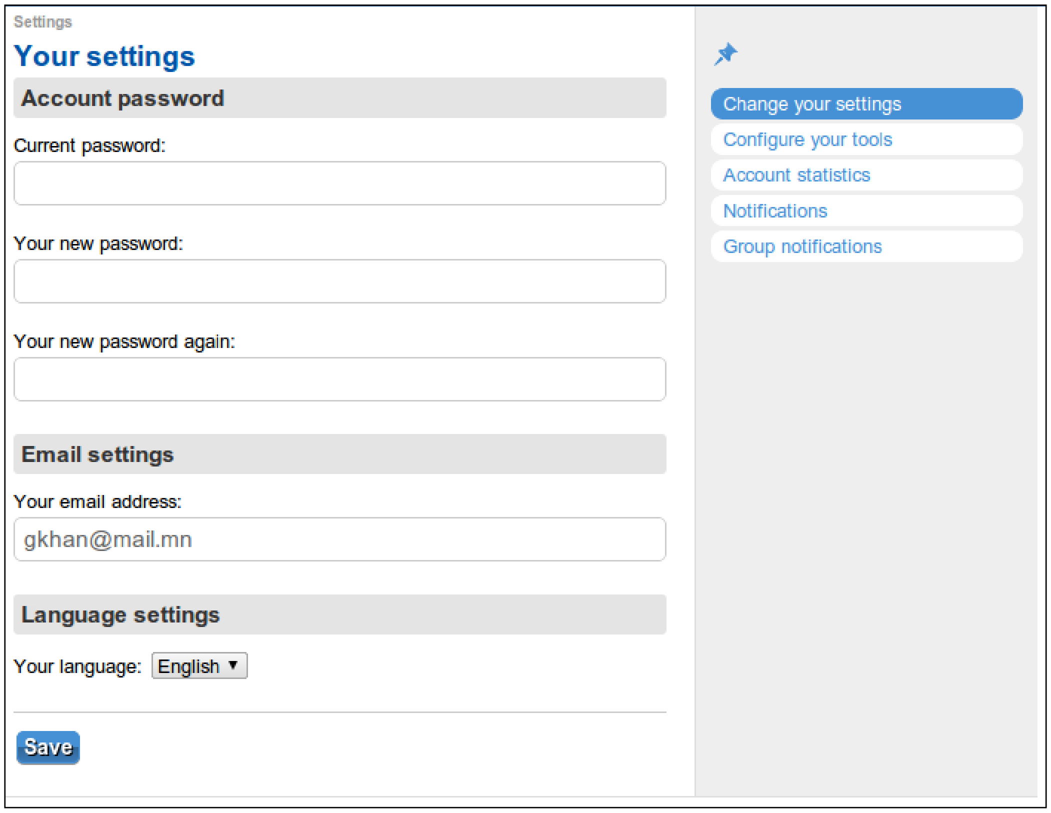
Who wants to go out for some good Indian food?

User settings

Users can configure their accounts in the **Settings** area of Elgg. A link to this area is always present in the topbar for easy access. Once in the settings area, users navigate to the various pages using the sidebar menu.

Your settings

The **Your settings** page has basic account options for e-mail address, password, and language. If an e-mail address is changed by a user, a validation e-mail is sent to ensure that the user owns the new e-mail address. English is the only language distributed with Elgg, but adding additional languages is discussed in *Chapter 6*.



Your settings

Account password

Current password:

Your new password:

Your new password again:

Email settings

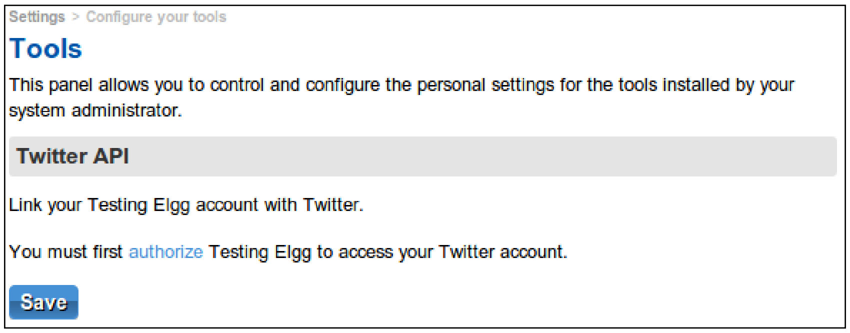
Your email address: gkhan@mail.mn

Language settings

Your language: English

Tools

The **Tools** page is a place where plugins can add configuration options. The only plugin distributed with Elgg that has a user-configurable setting is the Twitter API plugin. We describe in *Chapter 5* how to enable and configure it.



Settings > Configure your tools

Tools

This panel allows you to control and configure the personal settings for the tools installed by your system administrator.

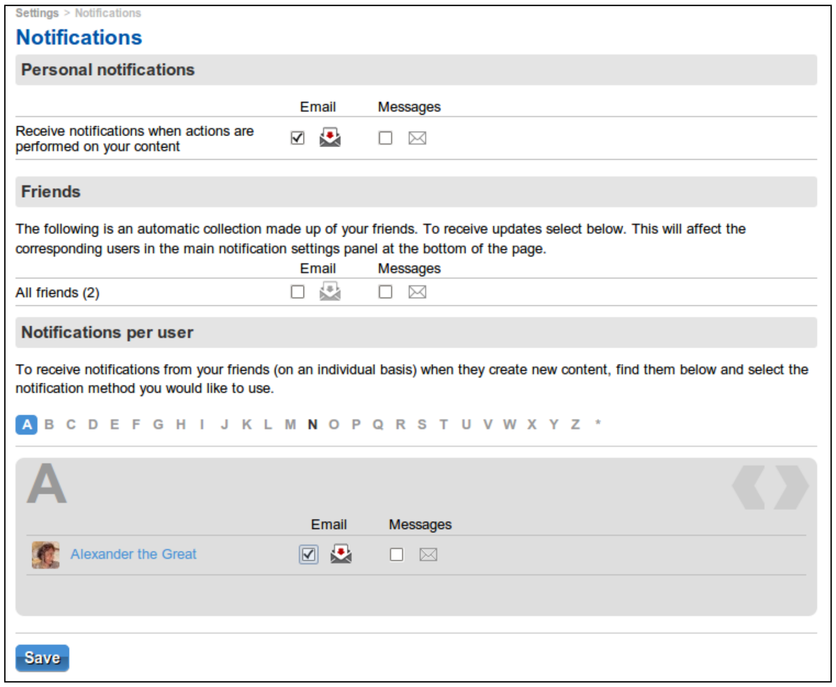
Twitter API

Link your Testing Elgg account with Twitter.

You must first authorize Testing Elgg to access your Twitter account.

Notifications

Users have a suite of notification settings available to them. Users can control what they are notified about and how they receive the notifications. These settings are divided into two pages: one for users and one for groups.



Settings > Notifications

Notifications

Personal notifications

	Email	Messages
Receive notifications when actions are performed on your content	✓	☐

Friends

The following is an automatic collection made up of your friends. To receive updates select below. This will affect the corresponding users in the main notification settings panel at the bottom of the page.

	Email	Messages
All friends (2)	☐	☐

Notifications per user

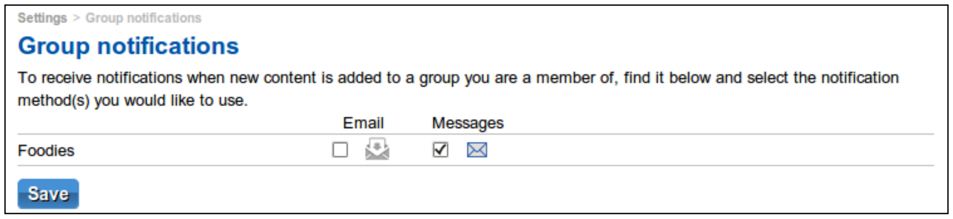
To receive notifications from your friends (on an individual basis) when they create new content, find them below and select the notification method you would like to use.

	Email	Messages
Alexander the Great	✓	☐

The first section of the user notifications page is for users to control how they receive notifications on their own content. For example, when a comment is left on one of their blogs, they can receive the notification through e-mail, through the private messaging tool, or not at all.

The next two sections deal with controlling notifications based on the activity of friends. By default, users do not receive notifications on the activity of their friends. To enable these notifications, users can activate notifications for all friends in the middle section or select individual friends at the bottom of the page.

The group notifications page works like the user notifications page. Group notifications are turned off by default just like friends-based notifications. When a user turns on notifications for a group, the user receives an e-mail or private message whenever there is a forum post or a user shares content with the group (such as creating a blog, bookmark, file, or page).


Settings > Group notifications

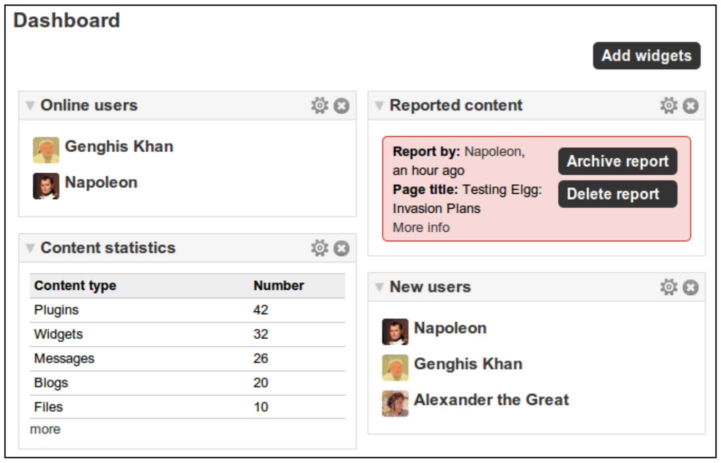
Group notifications

To receive notifications when new content is added to a group you are a member of, find it below and select the notification method(s) you would like to use.

	Email	Messages
Foodies	☐	☑

Administration

The administration area of Elgg is reached by clicking on the **Administration** link in the topbar. You entered this area after completing the installation of Elgg. The primary page is a dashboard that each administrator can customize with widgets.

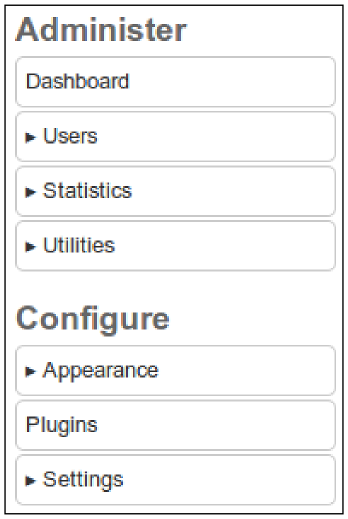


Report by: Napoleon, an hour ago
Page title: Testing Elgg: Invasion Plans
More info

Content type	Number
Plugins	42
Widgets	32
Messages	26
Blogs	20
Files	10

more

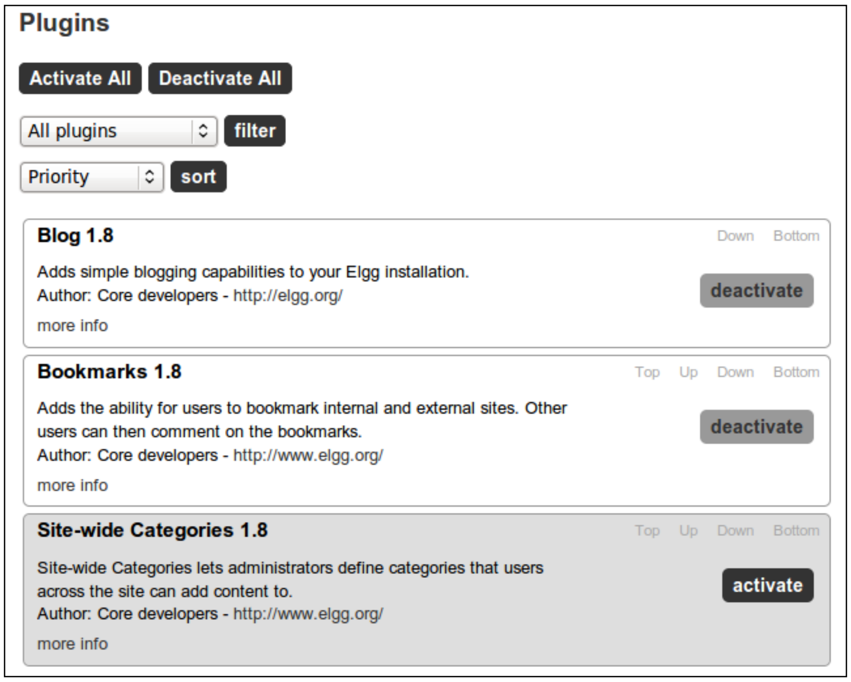
The sidebar menu is the primary mode of navigation. It is divided into two main sections: **Administer** and **Configure**. Pages that are used on a regular basis as an administrator monitors the site are in the Administer section. The **Configure** section has pages that deal with settings and plugins. An administrator uses these pages often when setting up a site, but only occasionally afterwards.



Now is a great time to familiarize yourself with the different menu items. For example, advanced site settings are located at **Configure | Settings | Advanced Settings**. On this page you can configure Elgg to run a private network, which means that only those who are logged into the site can see any content. If your site is on the Internet, you may want to turn off user registration while you test the site. This protects you from spammers joining the site during this testing phase. (*Chapter 10*, *Moving to Production* has more information on stopping spammers.) If you see a setting you do not understand, it is best not to modify it. Instead, try the convenient links in the footer to read documentation or interact with other people working with Elgg in the community forums.

Activating plugins

Most of Elgg's functionality is provided by plugins, which can be activated by going to **Configure | Plugins**. There is extensive information in *Chapter 6* about finding plugins, evaluating their quality, and configuring them. For now go to the plugin page and activate the site-wide categories plugin, which is covered next.



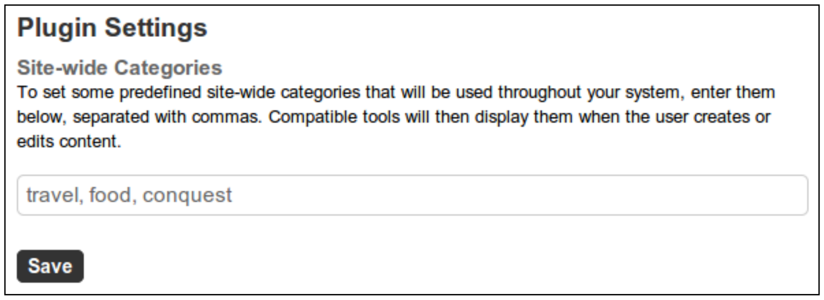
Blog 1.8 — Down Bottom
Adds simple blogging capabilities to your Elgg installation.
Author: Core developers - http://elgg.org/
more info
deactivate

Bookmarks 1.8 — Top Up Down Bottom
Adds the ability for users to bookmark internal and external sites. Other users can then comment on the bookmarks.
Author: Core developers - http://www.elgg.org/
more info
deactivate

Site-wide Categories 1.8 — Top Up Down Bottom
Site-wide Categories lets administrators define categories that users across the site can add content to.
Author: Core developers - http://www.elgg.org/
more info
activate

Site categories

Categories are set by an administrator for the entire site. Users can select these categories when they share content—whether blogs, bookmarks, files, or pages. The categories are defined through a comma-separated list of category titles.

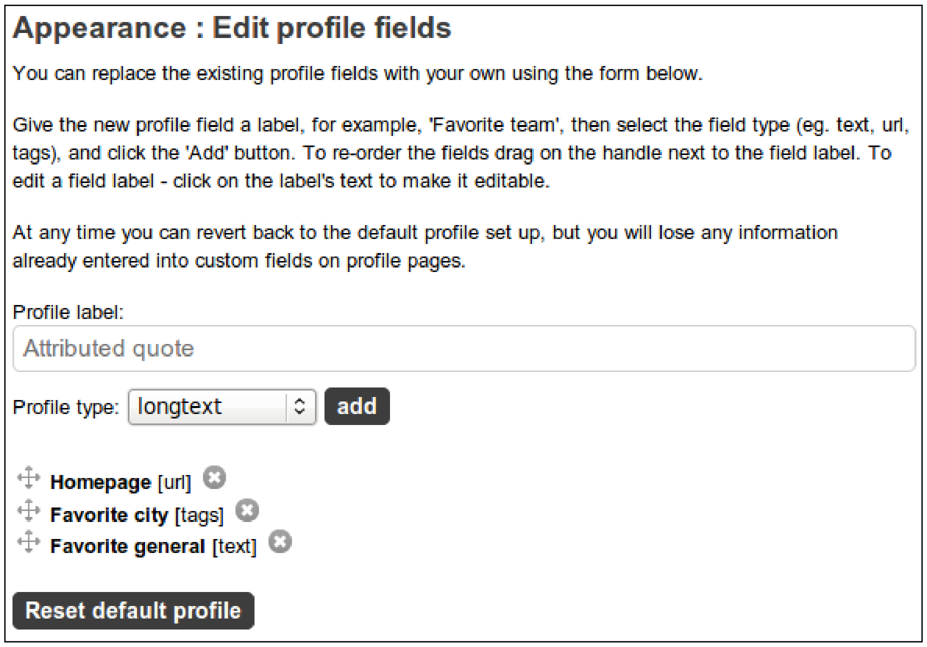


To set some predefined site-wide categories that will be used throughout your system, enter them below, separated with commas. Compatible tools will then display them when the user creates or edits content.

travel, food, conquest

Custom profile fields

The administrator can customize the profile fields available to users using the **Edit Profile Fields** page (**Configure | Appearance | Edit Profile Fields**). Each profile field has a label and a type. The type determines what kind of information can be entered for a profile field. To enter large amounts of text, select the longtext field. For a comma-separated list of tags, use the tags field. After creating a new set of profile fields, they can be ordered by dragging the arrow icon next to each field. *Chapter 6* includes a section on a community-contributed plugin that supports additional profile field types and allows administrators to group the fields into sections.


Appearance : Edit profile fields

You can replace the existing profile fields with your own using the form below.

Give the new profile field a label, for example, 'Favorite team', then select the field type (eg. text, url, tags), and click the 'Add' button. To re-order the fields drag on the handle next to the field label. To edit a field label - click on the label's text to make it editable.

At any time you can revert back to the default profile set up, but you will lose any information already entered into custom fields on profile pages.

Profile label:
Attributed quote

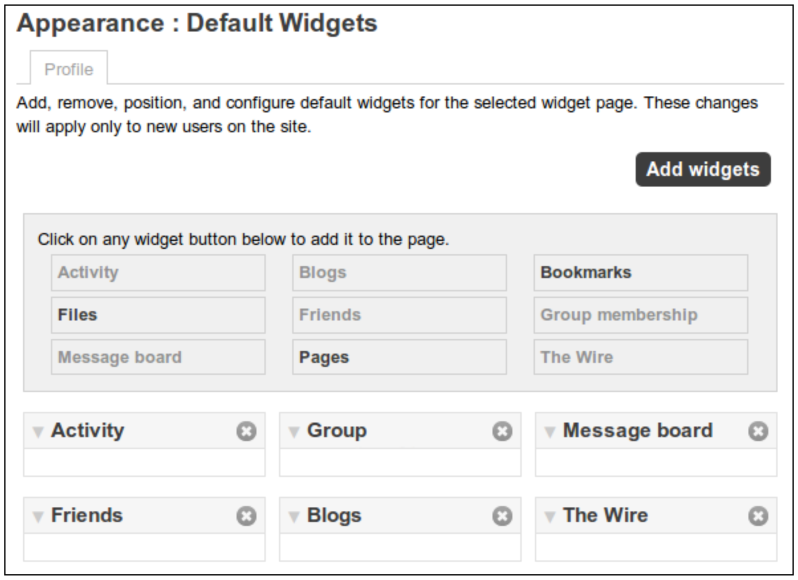
Profile type: longtext

Homepage [url]
Favorite city [tags]
Favorite general [text]

Default widgets

Your test users did not have any widgets on their profile pages when they logged in for the first time. This is an uninviting introduction to the site. To prevent this, administrators can set the initial widgets for a new user's profile page. This is done on the **Default Widgets** page available at **Configure | Appearance | Default Widgets**. The interface for selecting and arranging the widgets is the same drag-and-drop tool used on the profile page.

The best way to decide how to position the widgets is to try out different arrangements on your own profile to find what works. After the widgets are set on the administrative page, new users will get those widgets automatically when they register. Users that already existed on the site do not get the widgets.



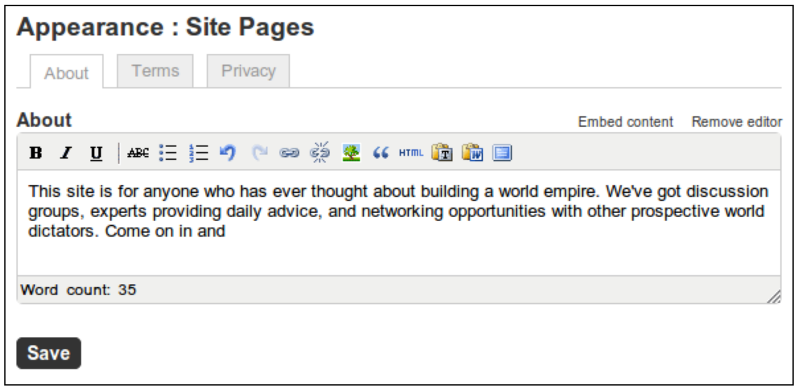
Appearance : Default Widgets

Add, remove, position, and configure default widgets for the selected widget page. These changes will apply only to new users on the site.

Click on any widget button below to add it to the page.

Site pages

One of the most requested features by Elgg administrators was a tool for creating static web pages. With the external pages plugin activated, administrators can create common web pages like an about page or a page with the site's privacy statement. To try this out, select the **Site pages** sidebar link under **Configure | Appearance** and use the editor to design a page.



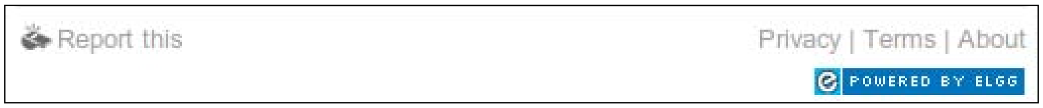
Appearance : Site Pages

About

This site is for anyone who has ever thought about building a world empire. We've got discussion groups, experts providing daily advice, and networking opportunities with other prospective world dictators. Come on in and

Word count: 35

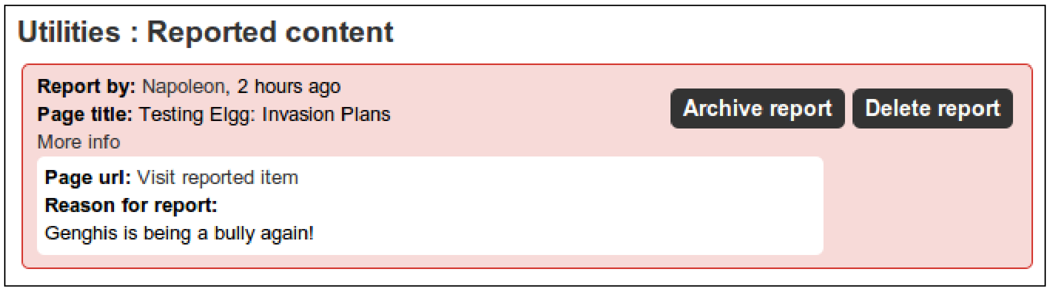
Links to the pages automatically appear in the site's footer as shown in the following screenshot:



Reported content

Sometimes users do things they should not do. This includes posting a photo that is not appropriate for the site or starting a flame war in a discussion forum. A busy site has too much activity for an administrator to keep track of everything that is happening. Fortunately, Elgg comes with a plugin that allows users to report inappropriate content to the administrator. On most pages, there is a whistle icon in the footer with a link labeled **Report this**.

When users click on the link, they fill out a description of the problem and submit the report. Administrators view the reports on the reported content page and can take whatever action is appropriate, including banning or deleting the user.


Utilities : Reported content

Report by: Napoleon, 2 hours ago
Page title: Testing Elgg: Invasion Plans
More info
Page url: Visit reported item
Reason for report:
Genghis is being a bully again!

Customizing your site

Now that you have completed a tour of your site, now is a great time to consider some possible design decisions. You probably have a set of requirements for your site or at least a general idea of how you would like it to work. As you worked through this chapter, you may have written down features that you would like to add or change to satisfy those requirements. If not, we have included a list of customizations that are often requested or discussed in the Elgg community forums. They can serve as examples to make you start thinking about your site. Many of them are already available as plugins that you can download and install.

Ideally, you should have a list of possible customizations ready before you read *Chapter 6*. That chapter describes the process of customizing an Elgg site by installing plugins. In it, you will learn how to find plugins, evaluate their quality, and install them on your site. If you are a developer (or are interested in learning a little web development), *Chapter 7*, *Creating Your First Plugin* and *Chapter 8*, *Customization Through Plugins* contain tutorials on building your own plugins.

User registration and authentication

- **Adding registration fields**: Additional fields can be added to the registration form. These fields can also be linked to the user profiles.
- **Import users**: A teacher may want to import a list of students to populate the accounts rather than having them manually sign up.
- **External authentication**: Plugins exist for using already existing usernames and passwords. Common services of this type include LDAP, ActiveDirectory, and Shibboleth.

Widgets

- **Fixing the widget layout**: Some site administrators want a consistent layout of profile widgets rather than giving the users control.
- **Push a new profile widget**: Rather than having users discover that a new widget has been added to the catalog, push the new widget to everyone's profile automatically.

Friendship model

- **Reciprocal friendships**: Instead of Elgg's one-way friendships, make all friendships two-way.
- **Confirmation**: Friend relationships are not formed until the person being "friended" confirms it.
- **Rename "friends" to "followers"**: On a corporate intranet site, "following" or "colleagues" terminology is usually more appropriate than "friends".

Roles

- **Moderators**: Special user accounts with the ability to edit or delete content without access to Elgg's administration area. Elgg is currently limited in its support for roles. There has been work on group-only administrators but very little progress in the area of site-wide moderators.

Help and support

- **Help pages**: Every site needs documentation on how to use the site as well as answers to common questions.
- **Feedback**: Give users the ability to ask questions, report problems, or offer suggestions to improve the site.

Summary

This chapter guided you on a tour of your Elgg website. Along the way you created test users, found friends, set up user profiles, adjusted settings, and used Elgg's administration area. You also got a taste of Elgg's major tools such as blogs and groups. These tools are the focus of the next two chapters. As we continue to explore Elgg's capabilities, further design hints and questions will be pointed out to prepare you for customizing your own site.

4
Sharing Content

People use social networking and social media sites to share things: photos, videos, family news, and more. This chapter covers the four major tools for sharing content that are distributed with Elgg:

- Blogs
- Bookmarks
- Files
- The Wire

We saw a quick overview of them in the previous chapter, but now we dig in and see what they can do. The tool reviews are broken down into four sections:

- Creating and uploading content
- Finding, viewing, and commenting on content
- Exploring use cases for the content sharing tool
- Possible customizations of the tool

A common mistake when designing and building a website is focusing on adding as many features as possible. A better approach is to develop short scenarios that explore how your users might interact with Elgg. These use cases help you focus on the features that matter, leading to a better design. The sample use cases included in this chapter are very short vignettes that relate to a particular application of Elgg (e-learning tool, organization intranet application, and niche social network). You can expand upon them or create your own that fit your audience and application.

There are many different ways to customize Elgg. To give you a sense of what is possible while designing your site, we will show several customizations that are commonly requested for these content sharing tools. All of them can be implemented through plugins. In some cases, the plugins are already written and are available in the Elgg community plugin repository. We cover finding and using plugins in *Chapter 6* so keep notes on possible customizations that you can refer to when working through that chapter.

Blogs

Whether users want to share their thoughts on the latest political scandal, describe a recent vacation, or announce important news to their friends, Elgg's blog tool makes it quick and easy. All it takes is filling out the title, writing the body, and clicking on **Publish**.

The features available from the blog plugin and the other bundled plugins include:

- WYSIWYG editor
- Saving drafts
- Preview a post
- Tags and categories
- Privacy control
- Embedding photos or links to files
- RSS feeds
- Commenting
- Liking blog posts

WYSIWYG

WYSIWYG stands for What You See Is What You Get. As you type and format the text on a WYSIWYG editor, you see what the text is going to look like when published. Without this kind of editor, you would need to mark up the text like this to achieve bold text:

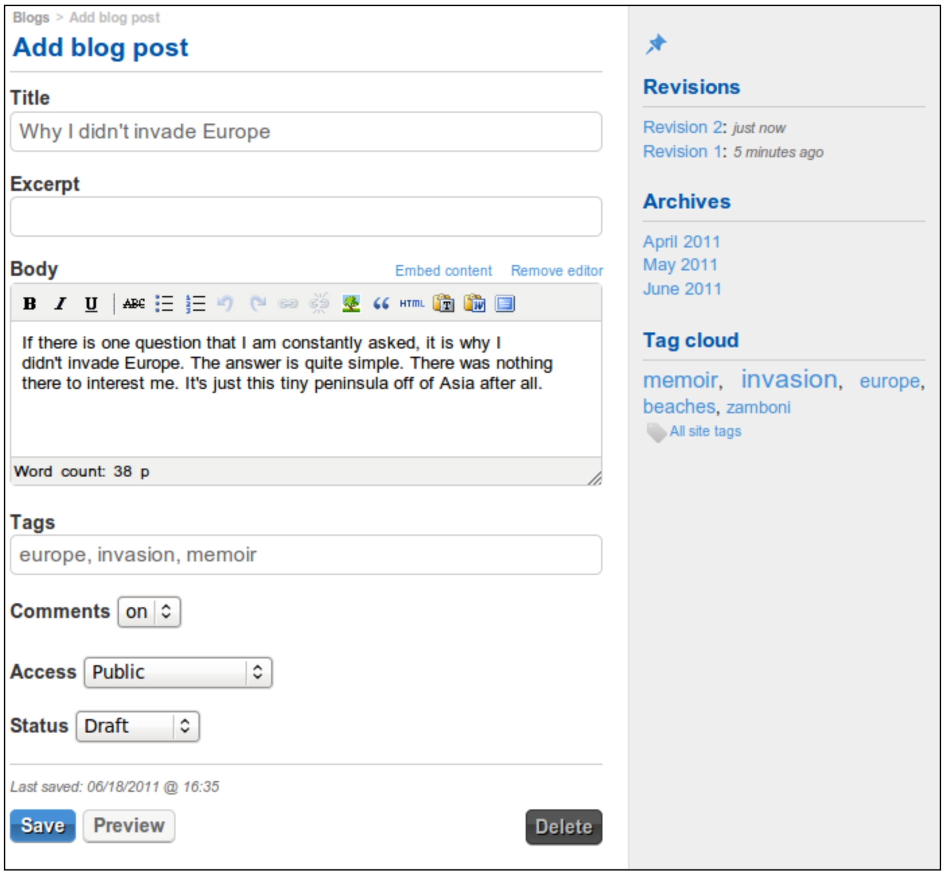
```
<b>This is bold text</b> and this is normal text.
```

With Elgg's editor it only takes a few clicks of the mouse to make text bold or italicized, create a link, or add a bulleted list. The editor is provided by the TinyMCE plugin distributed with Elgg. It is used throughout Elgg for editing and formatting text.

Creating a blog post

To create a blog post, select the **Blogs** tab from the site menu and click on the **Add blog post** button just under the site menu. You will see a form like that shown in the following screenshot. The title, an optional excerpt, body, and tags are entered in the main content area with controls below. If the categories plugin is turned on, the form also has check boxes for the categories that you set in *Chapter 3*.

The excerpt is displayed on pages that list many blog posts. The excerpt serves as a summary or teaser to encourage people to click on the link and read more. If a user does not enter text for the excerpt, Elgg uses the first few sentences of the post.



Title: Why I didn't invade Europe

Excerpt:

Body: If there is one question that I am constantly asked, it is why I didn't invade Europe. The answer is quite simple. There was nothing there to interest me. It's just this tiny peninsula off of Asia after all.

Word count: 38 p

Tags: europe, invasion, memoir

Comments: on

Access: Public

Status: Draft

Last saved: 06/18/2011 @ 16:35

Revision 2: just now
Revision 1: 5 minutes ago

April 2011
May 2011
June 2011

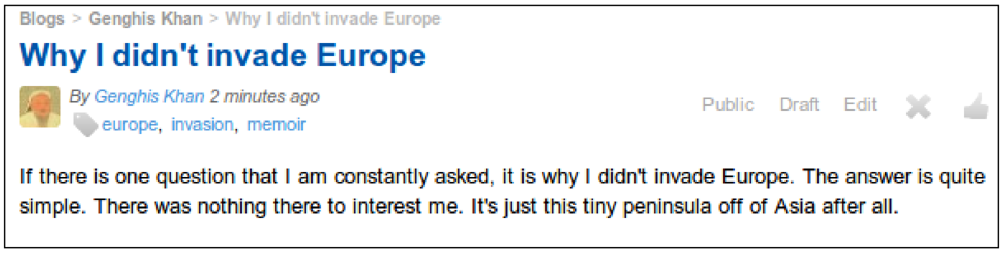
memoir, invasion, europe, beaches, zamboni

There are controls for setting the access level of the post, for determining whether comments are enabled, and for setting the post publication status.
A draft is automatically saved every five minutes while it is being written. A user can also set the status to **Draft** and click on the **Save** button to ensure that the latest changes are preserved.

Power users with HTML knowledge

Know HTML? You can click on the **Remove Editor** link above the body of the blog post and edit the raw HTML of the post.

For those who prefer seeing what a blog post will look like before publishing it, the post can be previewed while in draft state. This gives a user the opportunity to change some text or modify the formatting before making the post available to read.



Blogs > Genghis Khan > Why I didn't invade Europe

Why I didn't invade Europe

By Genghis Khan 2 minutes ago

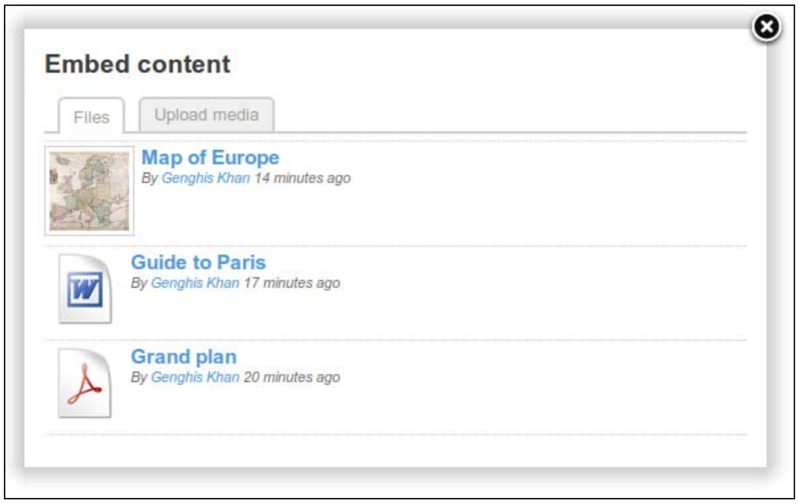
europe, invasion, memoir

Public Draft Edit

If there is one question that I am constantly asked, it is why I didn't invade Europe. The answer is quite simple. There was nothing there to interest me. It's just this tiny peninsula off of Asia after all.

Embedding photos and files

The WYSIWYG editor supports inserting images from the Internet, but what if a user wants to upload an image and embed it in the blog post? An **embed** plugin is distributed with Elgg that does just this. Click on the **Embed content** link above the body of a blog post and a dialog window pops up. This window has two tabs: one for files that have already been uploaded with the files tool and another for uploading new files.



Embed content

Files | Upload media

Map of Europe
By Genghis Khan 14 minutes ago

Guide to Paris
By Genghis Khan 17 minutes ago

Grand plan
By Genghis Khan 20 minutes ago

If an image is embedded, the image appears in the post. If another type of file is embedded, a link is inserted into that file's page. To change the size of the image or its position in the post, click on the image and then on the image button in the WYSIWYG editor's toolbar (it looks like a picture of a tree).

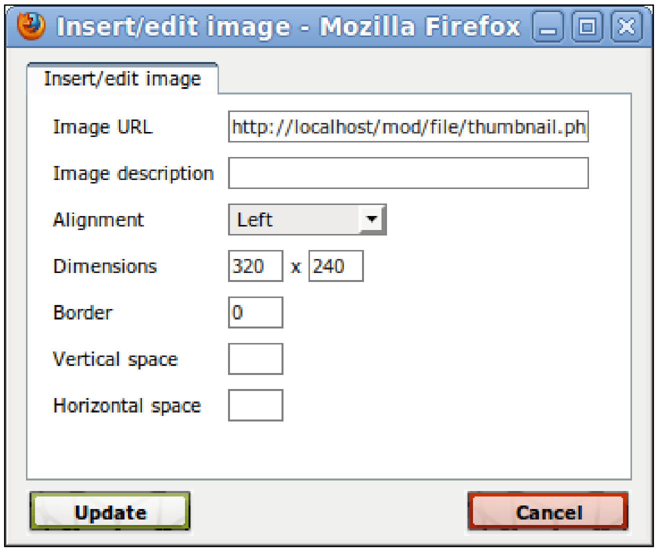


Image URL	http://localhost/mod/file/thumbnail.ph
Image description	
Alignment	Left
Dimensions	320 x 240
Border	0
Vertical space	
Horizontal space	

Publishing

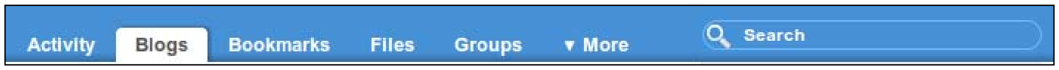
Publishing a blog post results in notifications being sent to the user's friends and an entry being added to the activity stream about the post.

Finding and viewing

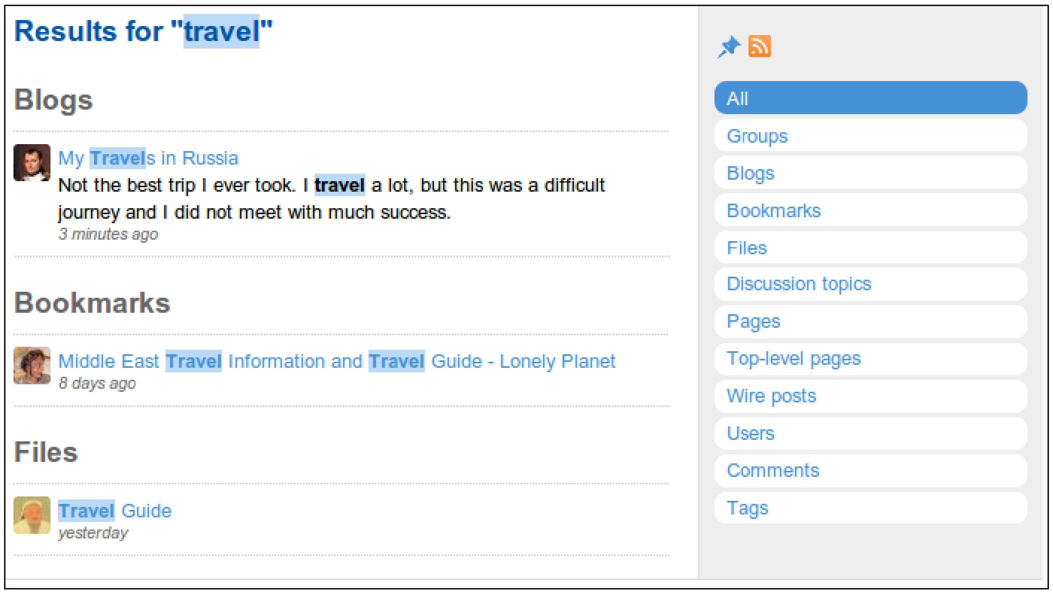
Users will not write many blog posts if people are not commenting on them. Before users can comment on a post, they have to know it exists. Notifications and activity streams are two ways for users to discover blog posts. Elgg also provides a list of recent posts, RSS feeds, profile widgets, and a search engine.

Search

The search box is on the far right of the topbar as shown in the following screenshot:



When a search query is entered, the search is performed over all the content on the site and the results are presented in a list broken down by content types. In the following example, the search results are divided by blog posts, bookmarks, and files:


Results for "travel"

Blogs

My Travels in Russia
Not the best trip I ever took. I travel a lot, but this was a difficult journey and I did not meet with much success.
3 minutes ago

Bookmarks

Middle East Travel Information and Travel Guide - Lonely Planet
8 days ago

Files

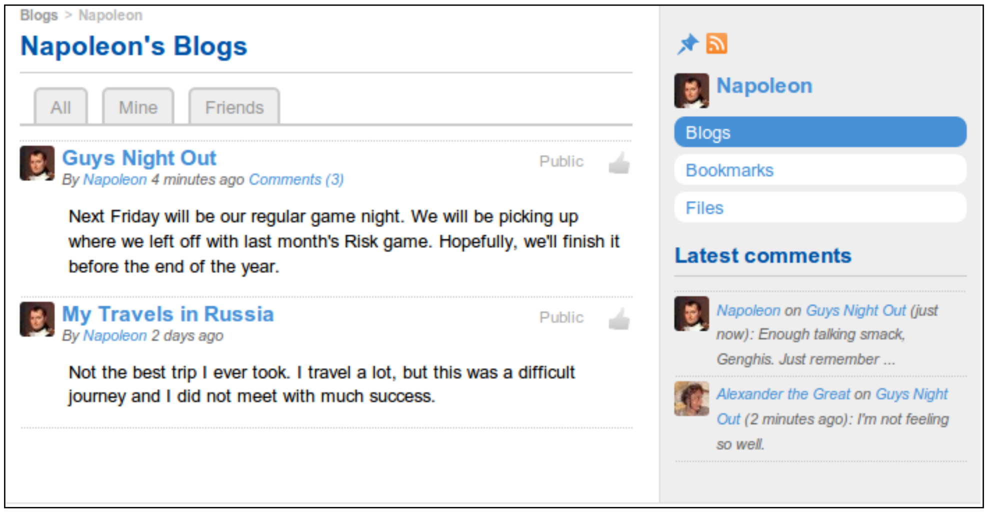
Travel Guide
yesterday

Blog posts with the search term in its title or body appear in the results. There is also a link at the bottom of the sidebar to select content that has been explicitly tagged with the search term.

Lists of blog posts

Users are taken to a list of their blog posts when they select **Blog** from the **Tools** menu. This page includes summaries of their recent blog posts with links to the full posts.

The following screenshot shows that the latest comments on the blog posts are included in the sidebar:



Guys Night Out — Public
By Napoleon 4 minutes ago Comments (3)

Next Friday will be our regular game night. We will be picking up where we left off with last month's Risk game. Hopefully, we'll finish it before the end of the year.

My Travels in Russia — Public
By Napoleon 2 days ago

Not the best trip I ever took. I travel a lot, but this was a difficult journey and I did not meet with much success.

Napoleon on Guys Night Out (just now): Enough talking smack, Genghis. Just remember ...

Alexander the Great on Guys Night Out (2 minutes ago): I'm not feeling so well.

Other lists available from the tabbed menu are blogs written by friends and all blogs on the site. Not shown in the preceding screenshot, blogs are also available in monthly archives.

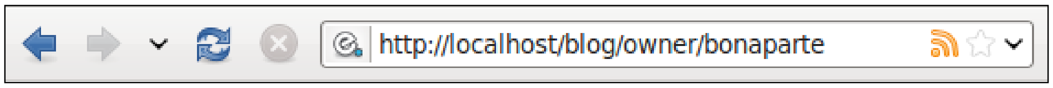
RSS feeds

Those lists are also available as RSS feeds. An RSS feed contains the latest 20 posts for a particular list. A user can follow an RSS feed by clicking on the orange RSS icon near the top of the sidebar.

What is RSS?

RSS (Really Simple Syndication) is a format for publishing regularly updated web content. The content is formatted to make it easy for computers to process and keep track of what has changed. Rather than continually checking a website using a web browser for new content, you can use software to tell you when there is new content and track what you have read. An example of such software is Google Reader, a popular web-based tool for reading RSS feeds.

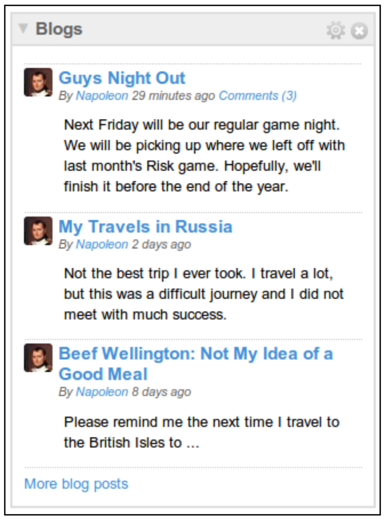
Most web browsers also have an orange icon that lights up when they detect an RSS feed in a page. Clicking on the icon subscribes the viewer to the feed. The following screenshot is an example of the orange icon in Firefox's address bar:



http://localhost/blog/owner/bonaparte

Widget

The blog tool provides a widget for users to put on their profiles. Users can control how many blogs are displayed through the widget's settings.



Blogs

Guys Night Out
By Napoleon 29 minutes ago Comments (3)

Next Friday will be our regular game night. We will be picking up where we left off with last month's Risk game. Hopefully, we'll finish it before the end of the year.

My Travels in Russia
By Napoleon 2 days ago

Not the best trip I ever took. I travel a lot, but this was a difficult journey and I did not meet with much success.

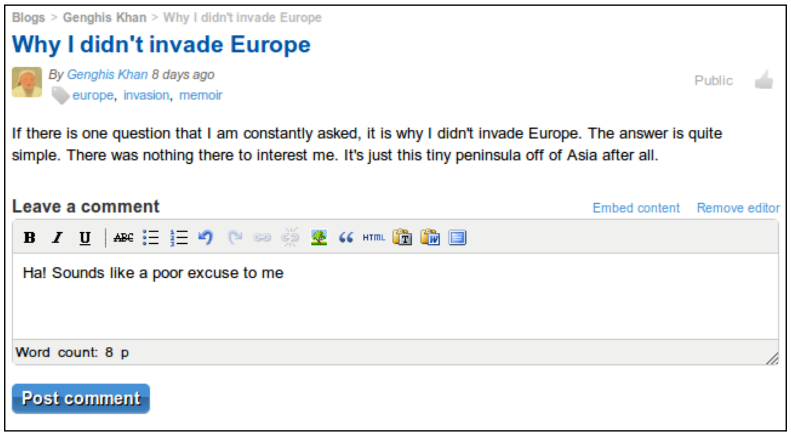
Beef Wellington: Not My Idea of a Good Meal
By Napoleon 8 days ago

Please remind me the next time I travel to the British Isles to ...

More blog posts

Commenting

Once users find interesting blog posts, they will want to interact with the author through comments. When a comment is posted, the author receives a notification of the comment (assuming the author has notifications enabled). The comment also appears in the activity stream, letting people know that this might be a blog post worth reading. Just like the list pages, an individual blog post page has an RSS feed. It contains the latest comments on the post.



Blogs > Genghis Khan > Why I didn't invade Europe

Why I didn't invade Europe

By Genghis Khan 8 days ago
europe, invasion, memoir

Public

If there is one question that I am constantly asked, it is why I didn't invade Europe. The answer is quite simple. There was nothing there to interest me. It's just this tiny peninsula off of Asia after all.

Leave a comment

Embed content Remove editor

Ha! Sounds like a poor excuse to me

Word count: 8 p

Post comment

Use cases

- **A communication tool for managers**: A manager uses a blog to keep her group updated on the internal news of the corporation. These blog posts do not require getting everyone into a room at the same time for a meeting and allow for public discussion in the comments area. The manager is also able to set the access level on her posts so that only members of her group can read them.
- **Book reports**: Students are writing persuasive essays. Each draft is a separate blog post giving the teacher and other students the opportunity to comment on the essays as they develop.
- **Site news**: The owner of a site writes about new features that will be rolled out and points out exciting activity on the site. The blog posts are featured prominently on the front page of the site. This not only provides an easy means of communicating with the users but also demonstrates one of the site's tools.

Customizations

When building a website with Elgg, most people want to add to or change the existing features provided by Elgg and its bundled plugins. We list a few common customizations below to get you thinking about your site. Most of these are already available as plugins in the Elgg community plugin repository. We discuss finding and evaluating these community plugins in *Chapter 6*. Custom plugins provided the greatest amount of control, tailoring Elgg to your needs.

- **Syndicate blog from external source**: Your users may already have blogs on other sites. They might want your site to republish their posts automatically.
- **Insert embed codes from YouTube**: Users may want to embed videos from video sites into a blog post. By default, Elgg strips these embed codes from posts as a security precaution. A plugin could accept these embed codes from particular trusted sites and filter out the rest.
- **Change the name from blog to poems**: On a poetry site, the blog tool works well as a poetry publishing platform. The name of the tool just needs to be changed. This is demonstrated in *Chapter 8*.
- **Moderate comments**: You may want to moderate comments or blog posts before they are published.

Bookmarks

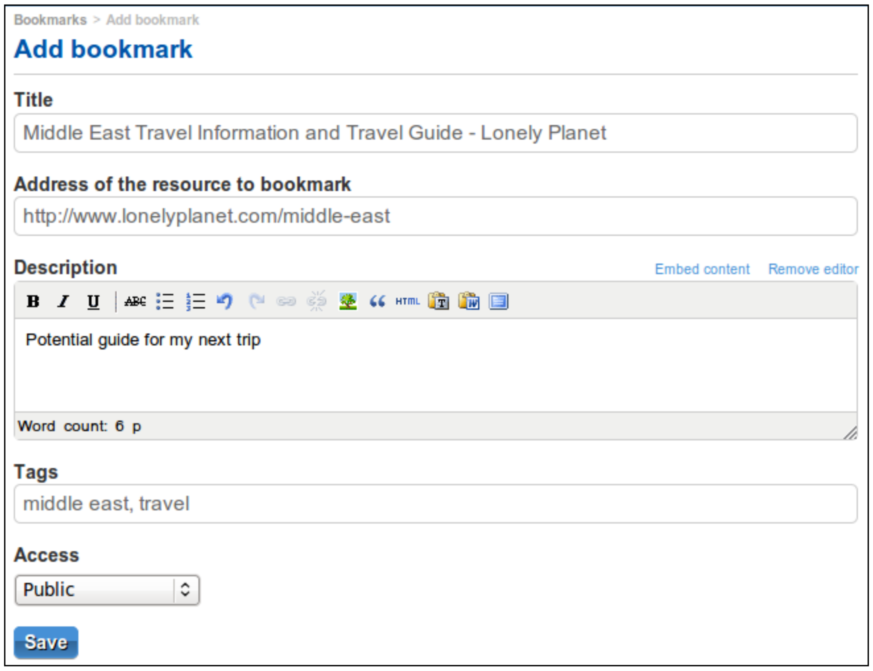
Upon finding an interesting website, many people do two things: bookmark it with their web browser and email it to their friends. Social bookmarking improves on this by storing the bookmarks on the web where they are available from any browser, not just the one used to bookmark the site. These social bookmarking websites also make it easy to share the bookmarks with other people. Elgg's social bookmarking tool works the same way so that users can save and share bookmarks.

Features offered by the bookmarks plugin and other bundled plugins include:

- Manual bookmark creation
- Bookmark icon for internal pages
- Bookmarklet
- Tags and categories
- Privacy control
- Commenting
- Liking

Adding a bookmark

On any of the bookmark list pages, there is a button in the top right of the content area for manually adding a bookmark. When creating a bookmark, a user enters a title, website address, an optional description of the site, any tags, and the access level. Before saving the bookmark, the user also has the opportunity to send the bookmark to friends. The friends are selected using the same tool that was used to turn on friend notifications in *Chapter 3*. When the bookmark is saved, it appears in the inboxes of the selected friends. Notifications are also sent and the activity stream is updated.



Add bookmark

Title

Middle East Travel Information and Travel Guide - Lonely Planet

Address of the resource to bookmark

http://www.lonelyplanet.com/middle-east

Description

Potential guide for my next trip

Word count: 6 p

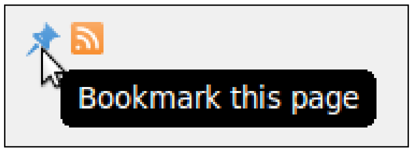
Tags

middle east, travel

Access

Public

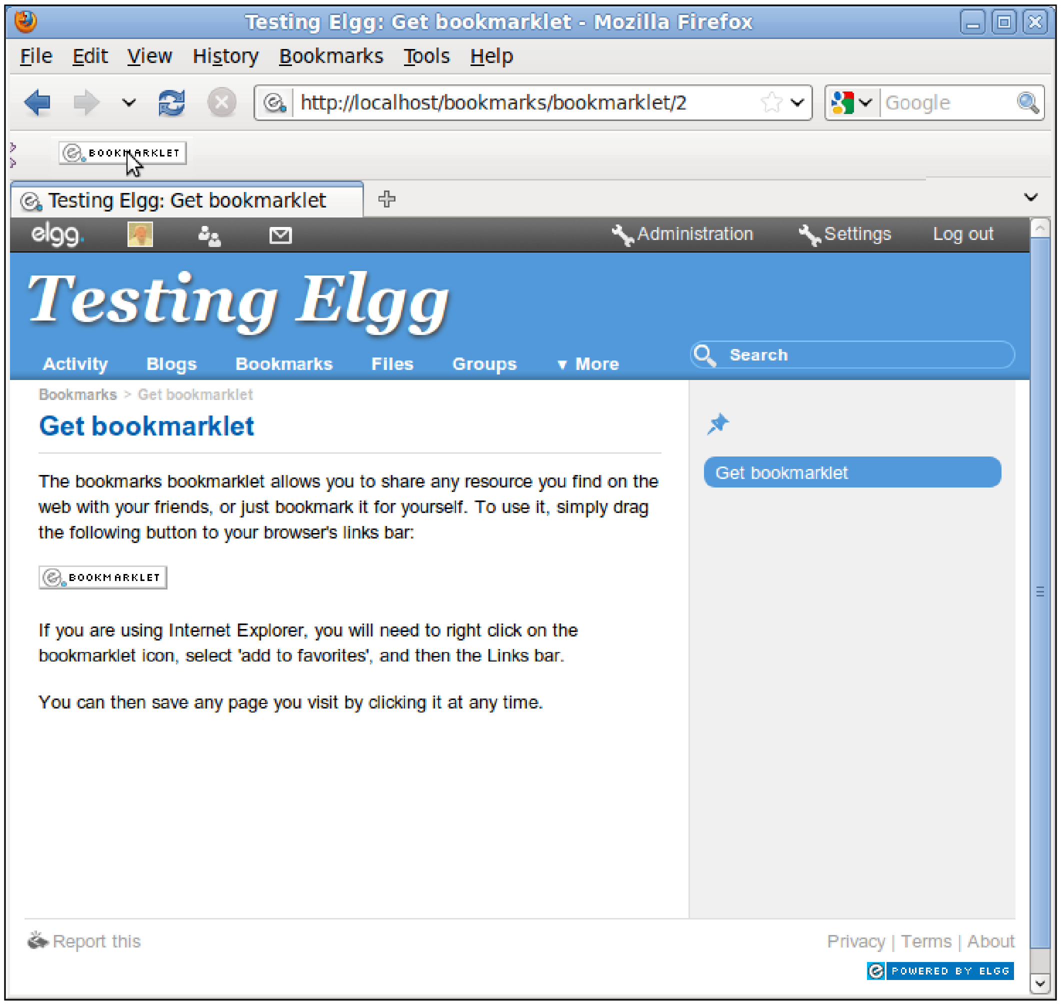
You may have noticed a push pin icon at the top of the sidebar. When a user hovers over the icon, a tool tip appears as shown in the following screenshot. (The visual appearance of the tool tip depends on the browser and desktop theme).



When a user clicks on this icon, the page being viewed is bookmarked. Elgg fills out the title and the address for the user. This makes it easy for users to store interesting pages from your site and share them with others.

Bookmarklet

A little known feature of the bookmarks tool is its bookmarklet. This is a button that can be dragged from the **Get bookmarklet** page to the bookmarks toolbar of a user's web browser. When visiting an interesting site, a user can click on the button and be taken to the Elgg bookmark creation page with the title and address of the site already filled out.



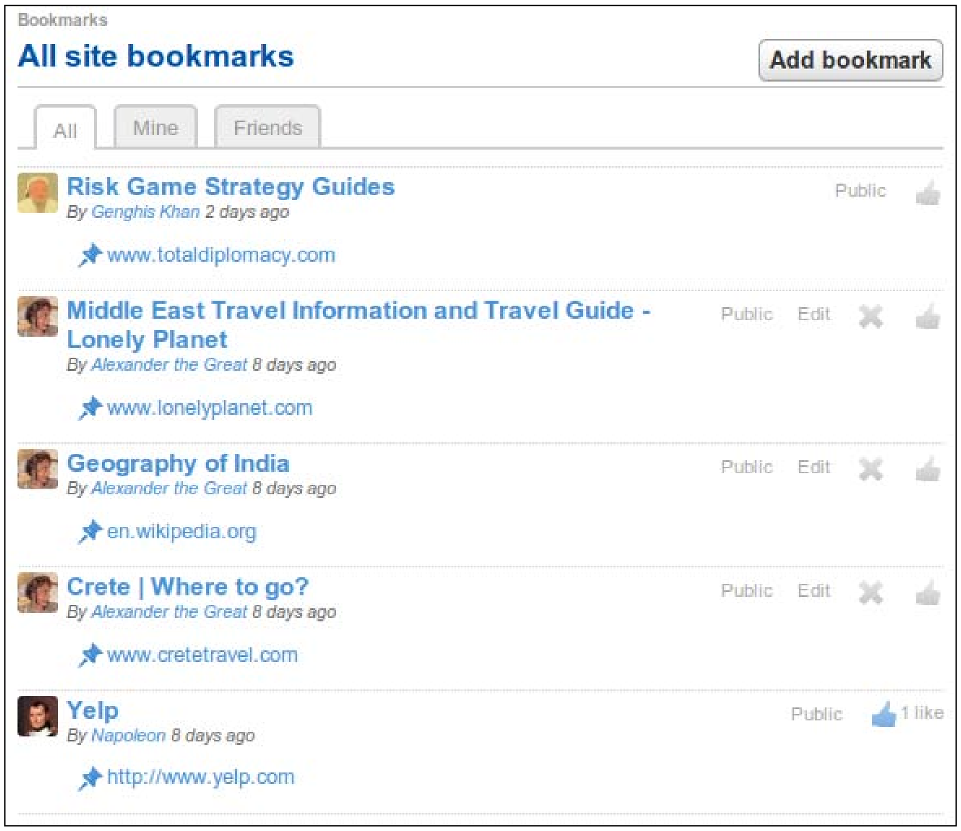
The bookmarks bookmarklet allows you to share any resource you find on the web with your friends, or just bookmark it for yourself. To use it, simply drag the following button to your browser's links bar:

If you are using Internet Explorer, you will need to right click on the bookmarklet icon, select 'add to favorites', and then the Links bar.

You can then save any page you visit by clicking it at any time.

Viewing

The bookmarks tool provides viewing functionality similar to the blogs tool. Users get lists of their bookmarks, their friends' bookmarks, and all the bookmarks on the site. Each of these lists has an RSS feed so that users can stay updated on the latest bookmarks.



Risk Game Strategy Guides
By Genghis Khan 2 days ago
www.totaldiplomacy.com
Public

Middle East Travel Information and Travel Guide - Lonely Planet
By Alexander the Great 8 days ago
www.lonelyplanet.com
Public Edit

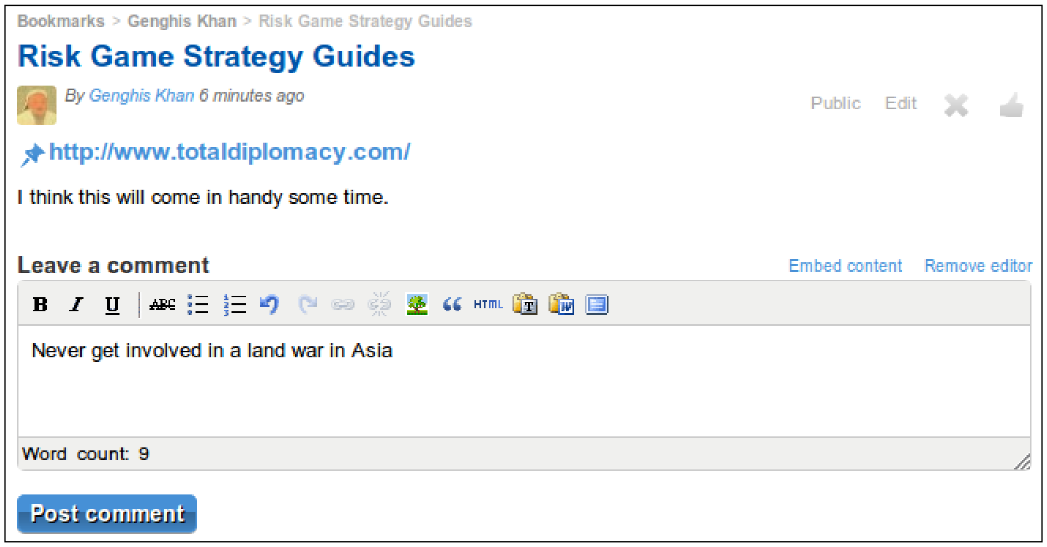
Geography of India
By Alexander the Great 8 days ago
en.wikipedia.org
Public Edit

Crete | Where to go?
By Alexander the Great 8 days ago
www.cretetravel.com
Public Edit

Yelp
By Napoleon 8 days ago
http://www.yelp.com
Public 1 like

There is also a widget so that users can display their latest bookmarks on their profiles. Just like the blog widget, the number of bookmarks displayed is a widget setting.

The page for displaying a bookmark looks similar to a blog post page except that there is a large link to the bookmark above the description text.


Bookmarks > Genghis Khan > Risk Game Strategy Guides
Risk Game Strategy Guides
By Genghis Khan 6 minutes ago
Public
Edit
http://www.totaldiplomacy.com/

I think this will come in handy some time.

Leave a comment
Embed content
Remove editor

Never get involved in a land war in Asia

Word count: 9
Post comment

Use cases

- **Literature or vendor search**: Your team at work has been assigned to find the best software package for managing an inventory. Rather than emailing around links, the team uses the Elgg bookmarklet to quickly bookmark and share vendor sites with each other. The comment sections are used to discuss features, advantages, and drawbacks before the information is summarized in a formal report.
- **Students bookmarking resources for reports**: Each student has been assigned a country and is asked to write a report about an important news story in that country. The students are asked to contrast the local coverage of the story with the world coverage. The first step is finding news resources on the Web. While bookmarking the websites, the students use the description field to summarize the content and the comment area to provide commentary, creating an online annotated bibliography in the process.

File

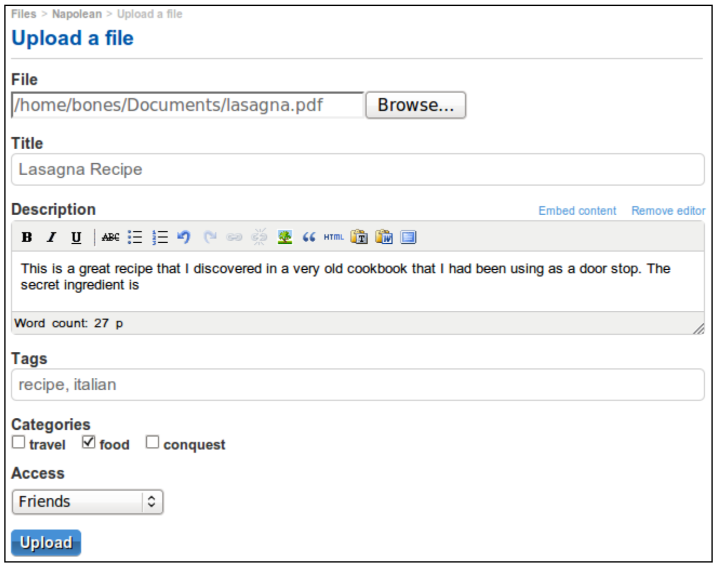
Users can share any kind of file with the file tool. This provides a simple interface for uploading and describing files. It is also integrated with Elgg's notifications, activity stream, and commenting system.

The file plugin and the other bundled plugins provide these features:

- Supports any file type
- Audio player
- Tags and categories
- Privacy control
- Commenting
- Liking

Uploading a file

Before uploading a file, a user fills out information about the file. The file is selected from the user's computer using the web browser's file chooser. When the user clicks on the **Upload** button, the file is uploaded to the server. This can take some time depending on the size of the file and the speed of the Internet connection. Once the upload is complete, Elgg stores the file on the server and assigns an icon to it based on the file type. Notifications are sent out to the user's friends and the activity stream is updated with information about the new file.


Files > Napolean > Upload a file

Upload a file

File: /home/bones/Documents/lasagna.pdf

Title: Lasagna Recipe

Description: This is a great recipe that I discovered in a very old cookbook that I had been using as a door stop. The secret ingredient is

Word count: 27 p

Tags: recipe, italian

Categories: travel [✓ food] conquest

Access: Friends

White Screen of Death

If you upload a file that is too large based on your server's configuration, you will be greeted with a White Screen of Death (WSOD). When PHP runs out of memory, it displays a message about a fatal error and then quits. Elgg is configured to hide these errors by default as they can be confusing to users. The size of files that you want to support will influence how much memory you need to give PHP. *Chapter 10* discusses this in more detail.

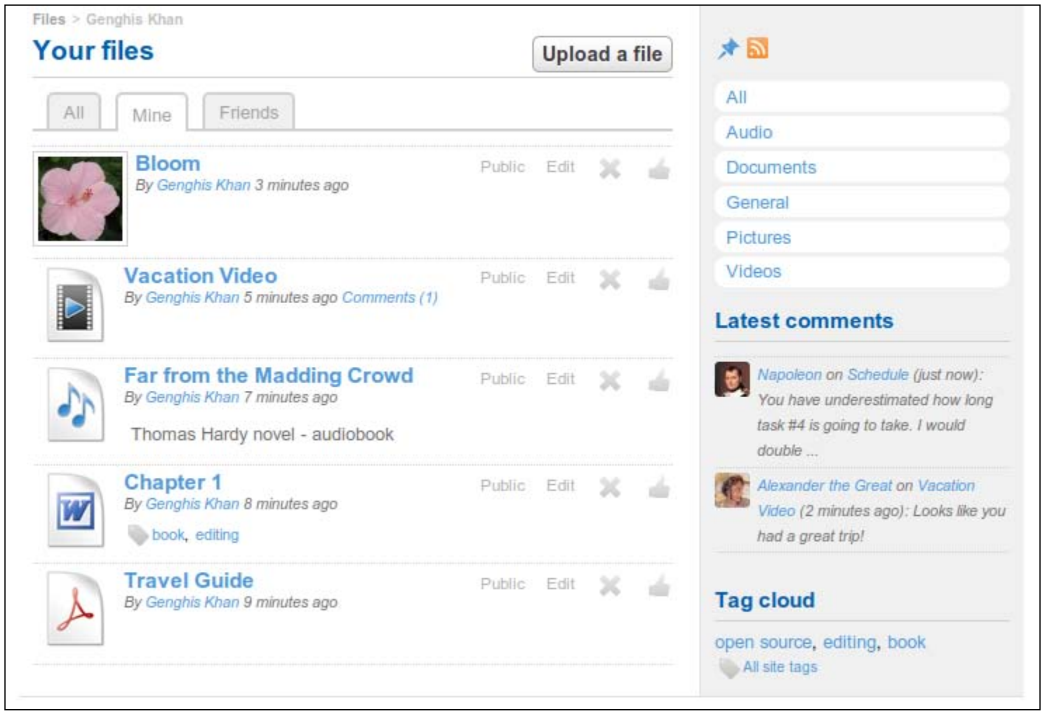
If a user wants to update the file later, the user clicks the edit link on the file's page and selects a new version of the file to upload. Notifications are not sent when a file is updated.

Is this secure?

Allowing users to upload files to your server can be a dangerous proposition. If the user can upload a script and execute it, all sorts of damage can be caused. Fortunately, Elgg does this the correct way by storing the files in a special data directory. Users cannot remotely run any file stored there, which prevents a malicious user from uploading and executing a script. This is a big security advantage that Elgg has over many other web applications that allow file uploads.

Viewing

The file tool provides the same list pages and a profile widget as the other tools. The list pages include one for a user's files, one for the user's friends, and one for all files on the site. In addition, Elgg automatically creates lists categorized by file type to organize the files. As you would expect, all of the different list pages provide RSS feeds.



Bloom
By Genghis Khan 3 minutes ago

Vacation Video
By Genghis Khan 5 minutes ago Comments (1)

Far from the Madding Crowd
By Genghis Khan 7 minutes ago
Thomas Hardy novel - audiobook

Chapter 1
By Genghis Khan 8 minutes ago
book, editing

Travel Guide
By Genghis Khan 9 minutes ago

Latest comments

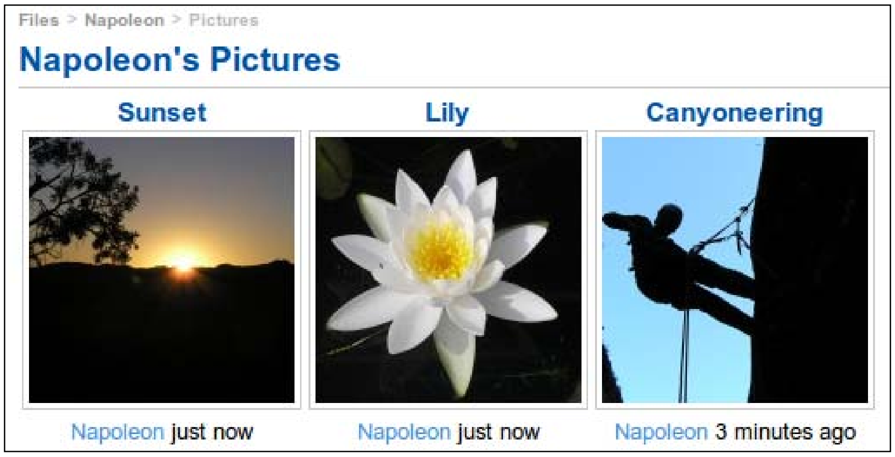
Napoleon on Schedule (just now): You have underestimated how long task #4 is going to take. I would double ...

Alexander the Great on Vacation Video (2 minutes ago): Looks like you had a great trip!

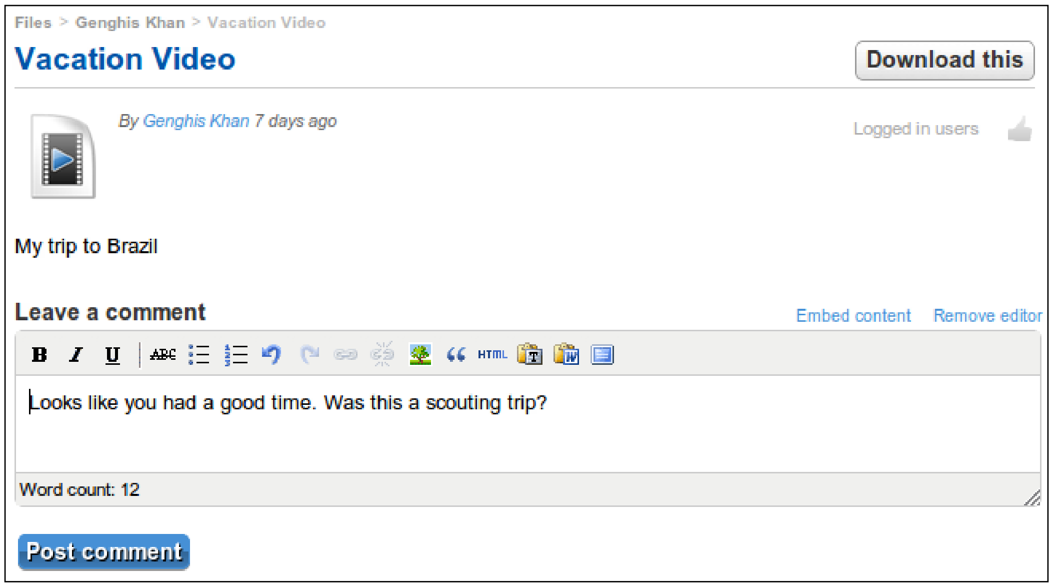
Tag cloud

open source, editing, book

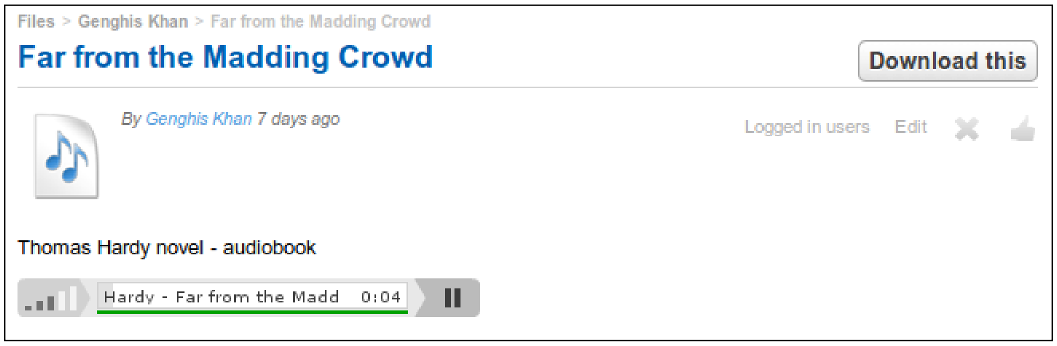
When selecting to view just pictures, the file tool presents them in a gallery view. This is much more visually appealing than a list of file names with small icons.



The page for viewing an individual file should look familiar by now. It includes the title, description, tags, and a comment area. It also contains the **Download this** button so that users can access the file.



Just as photos have a special gallery view, there is a special feature for audio files. When you enable the Zaudio plugin, a Flash-based audio player is inserted on audio file pages so that people can listen to the files in their web browser.


Files > Genghis Khan > Far from the Madding Crowd
Far from the Madding Crowd
Download this
By Genghis Khan 7 days ago
Logged in users
Edit
Thomas Hardy novel - audiobook
Hardy - Far from the Madd 0:04

Use cases

- **Sharing code snippets and scripts**: Software developers usually have scripts and snippets of code lying around that perform common tasks like renaming every file in a directory. A developer would not package those scripts into a library to give to others because they seem disposable. What if there was a simple mechanism to share these bits of code across an organization? Using the file tool, developers could upload, tag, and comment on scripts and snippets.
- **Maps and trail guides for a hiking group**: There could be a hiking group on a general purpose site or an entire site dedicated to hiking with location specific groups. The users would upload electronic versions of maps and trail guides for local hikes. A rating tool could be incorporated to make it easy to find the best hikes.

Customizations

Here are a few sample customizations that can be implemented through plugins. *Chapter 6* describes the process of finding plugins in the Elgg community plugin repository. *Chapter 7* and *Chapter 8* provide tutorials on building plugins.

- **User quotas**: The hard drive on your server does not have infinite space.
- **Folders**: If users are uploading many files, being able to organize them into folders is a great feature.
- **Video playing**: If you can play MP3 files in the browser, why not video too?
- **Limiting file types**: You may want to only allow Word documents and Excel spreadsheets for your site.

The wire

You may not know what microblogging is, but you have likely heard of the popular microblogging site, Twitter. Microblogging is simply a very short blog post, usually limited to 140 characters. People use this format for sharing updates on what they have been doing, commenting on news events, or posting links to websites. The Elgg microblogging tool is called The Wire and works much like Twitter.

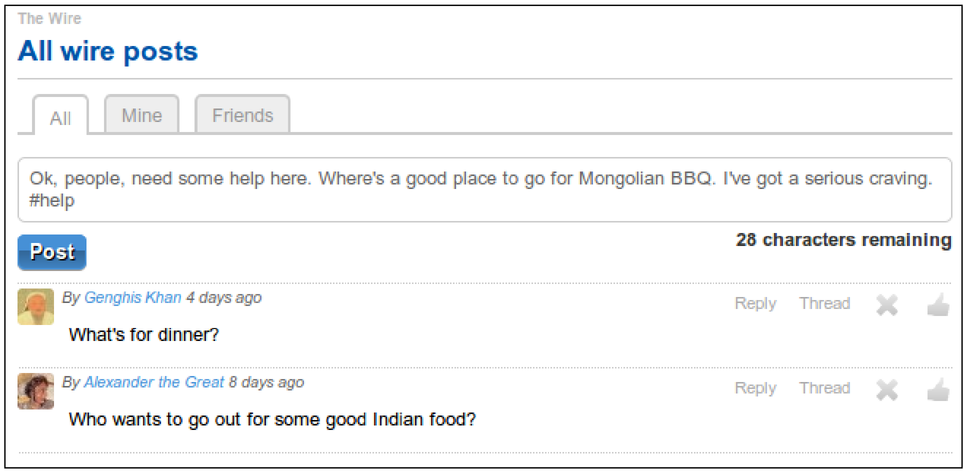
The features of the wire plugin include:

- 140 character posts
- Hashtags
- Replies

- Conversation threads
- RSS feeds

Posting

Selecting **The Wire** from the site menu takes users to a page where they can read the latest wire posts or add their own. As a user types a post, a counter underneath the text box indicates how many characters are left. When a post is saved, notifications are sent to the poster's friends and the activity stream is updated.



Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

28 characters remaining

By Genghis Khan 4 days ago

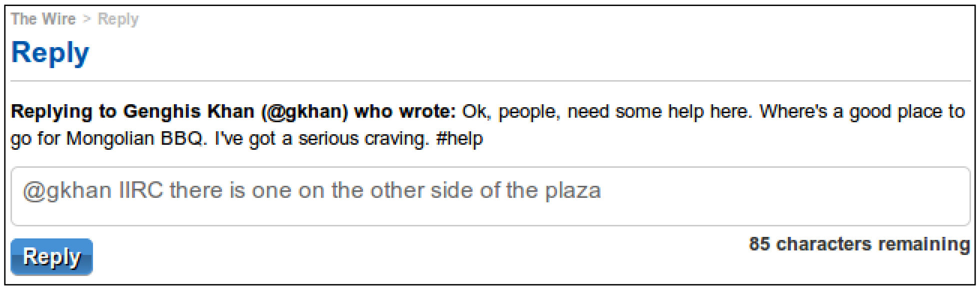
What's for dinner?

By Alexander the Great 8 days ago

Who wants to go out for some good Indian food?

The wire automatically changes web addresses, e-mail addresses, hashtags, and usernames to links when the wire posts are displayed. A hashtag (like #help in the preceding screenshot) becomes a link to all the posts with that tag. A username preceded by the @ sign (such as `@cash`) links to that user's wire page.

Clicking the reply link on a wire post takes a user to a page that displays the original post with the poster's username to make it easy to include it in the message. The original poster is notified of the new post even if they are not friends.

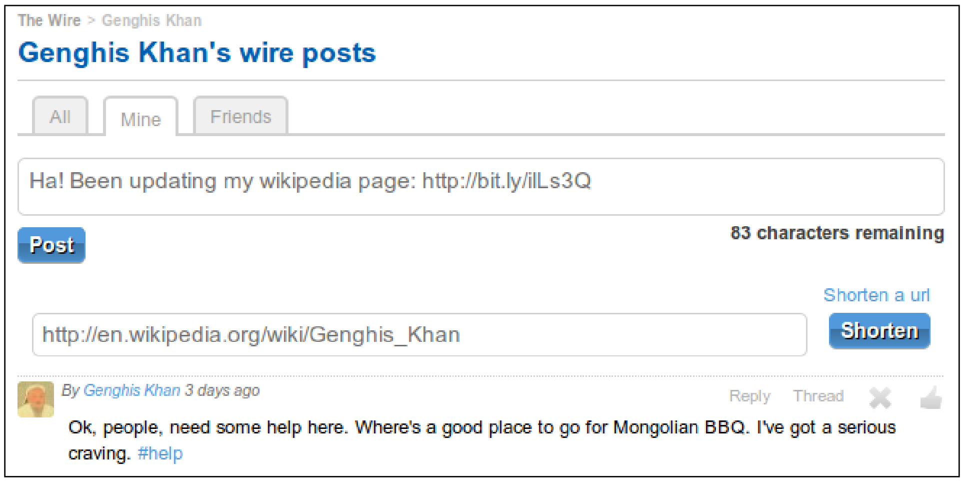


Replying to Genghis Khan (@gkhan) who wrote: Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

@gkhan IIRC there is one on the other side of the plaza

85 characters remaining

Posting a link to an article on the Web along with a personal reaction is very common in microblogging. It is challenging to say much of anything though with the link taking up most or all of the 140 characters. To work around this limitation, many use URL shortening services like TinyURL or `bit.ly`. These services create a short URL such as `http://tinyurl.com/qdsq92` that redirects people to the original website when clicked. The wire includes a hook so that a URL shortener can be added to the wire page through a plugin. Users can then skip the process of going to a site like TinyURL to get the shortened link.


The Wire > Genghis Khan

Genghis Khan's wire posts

Ha! Been updating my wikipedia page: http://bit.ly/ilLs3Q

83 characters remaining

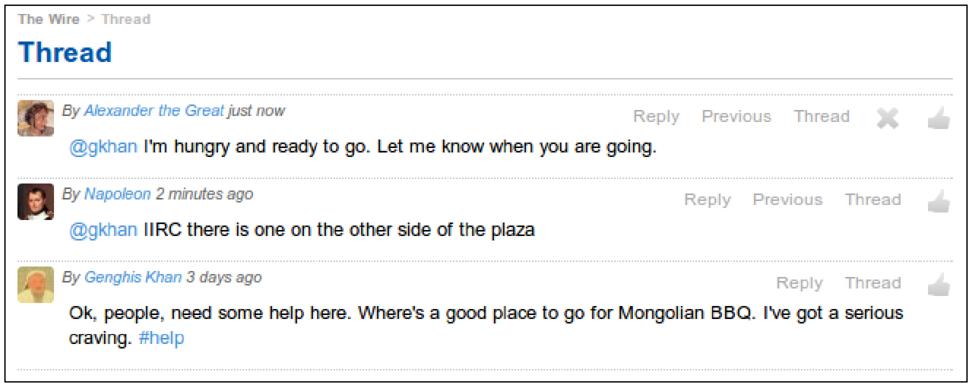
http://en.wikipedia.org/wiki/Genghis_Khan

By Genghis Khan 3 days ago

Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

Viewing

Besides the **All**, **Mine**, and **Friends** pages common to all the tools, the wire provides a look at a conversation thread. This page displays a post and all the posts that were made in reply to it and all the replies to those replies and so on. This is done in chronological order so that users can read through the conversation thread.


The Wire > Thread

Thread

By Alexander the Great just now

@gkhan I'm hungry and ready to go. Let me know when you are going.

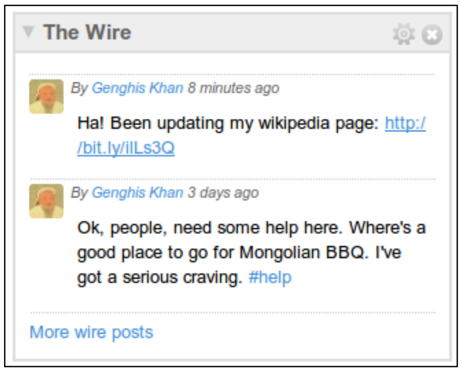
By Napoleon 2 minutes ago

@gkhan IIRC there is one on the other side of the plaza

By Genghis Khan 3 days ago

Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

The wire also has a profile widget that allows a user to select the number of posts to display.



By Genghis Khan 8 minutes ago

Ha! Been updating my wikipedia page: http://bit.ly/ilLs3Q

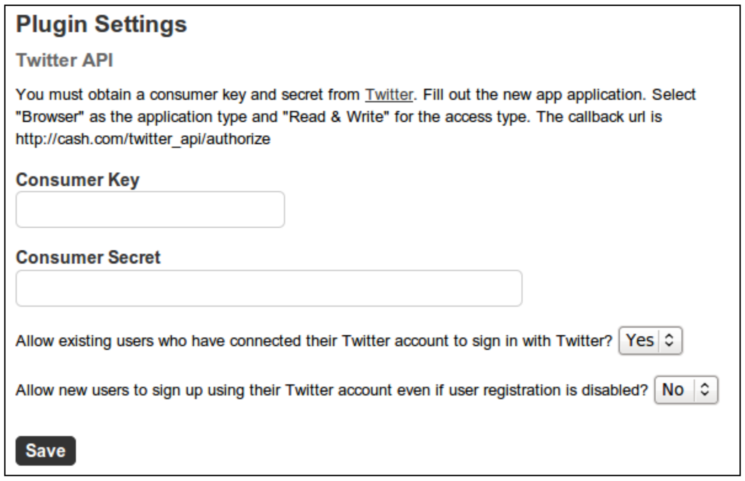
By Genghis Khan 3 days ago

Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

Twitter integration

Elgg is distributed with a Twitter API plugin. It handles posting users' wire posts to their Twitter accounts. This requires that an administrator register the Elgg website with Twitter and the users must authorize the Elgg site to post tweets to their Twitter account.

To activate the plugin, go to the the **Plugin** page in the administration area and activate the OAuth API plugin. (This is an example of one plugin depending on another and we will cover Elgg's plugin dependency system in *Chapter 6*.) Next, activate the Twitter API plugin and then click the plugin's settings link. An Elgg site needs two keys from Twitter before Twitter will accept data from it. The instructions, as shown in the following screenshot, include a link to Twitter's app registration page:



You must obtain a consumer key and secret from Twitter. Fill out the new app application. Select "Browser" as the application type and "Read & Write" for the access type. The callback url is http://cash.com/twitter_api/authorize

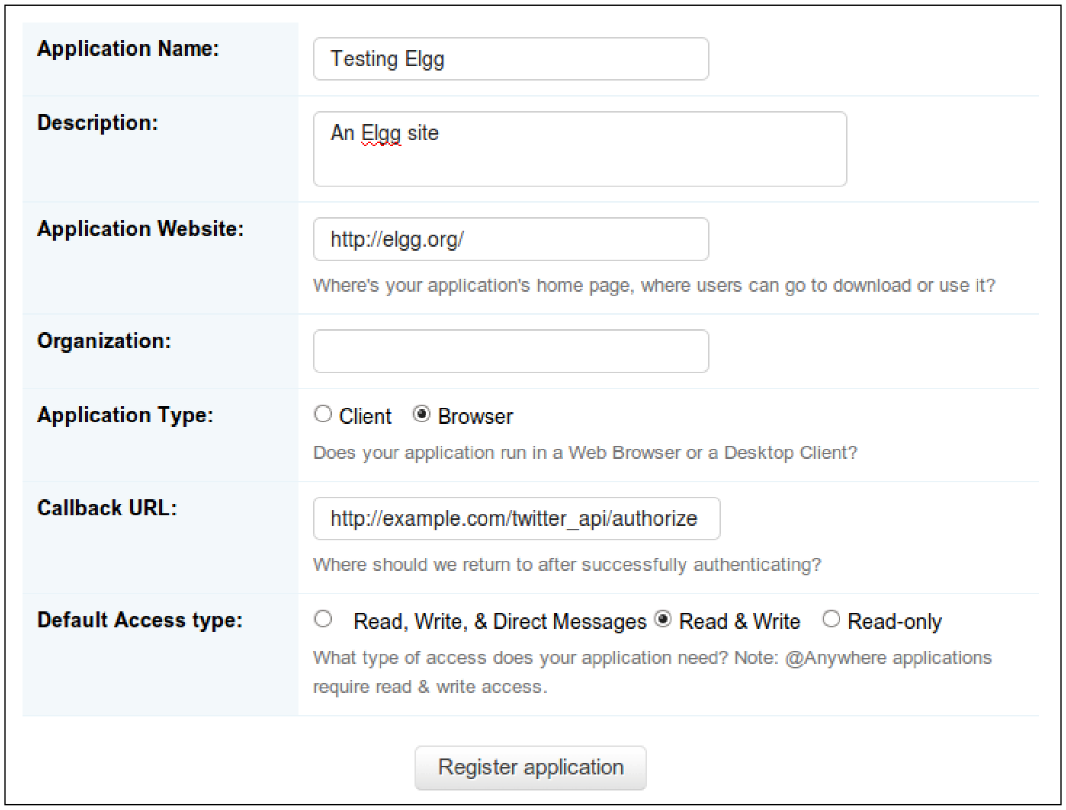
Consumer Key

Consumer Secret

Allow existing users who have connected their Twitter account to sign in with Twitter? Yes

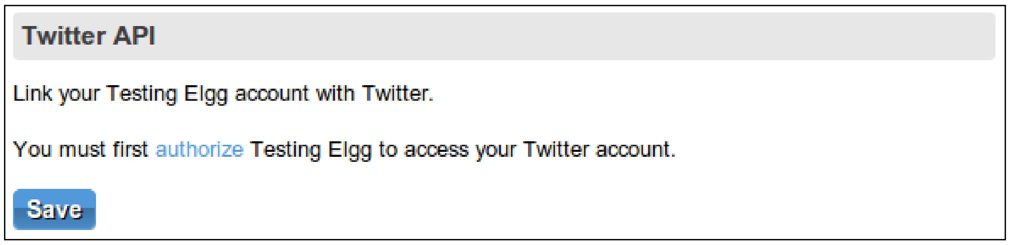
Allow new users to sign up using their Twitter account even if user registration is disabled? No

Twitter's registration page asks for several details about the Elgg website. The **Application Name** is displayed to users when they authorize the Elgg site to act on their behalf. In order to post new tweets, the application must have both read and write access. Finally, the callback URL must be set. This is where Twitter sends the user after approving the application's request for access. The instructions on the Twitter API settings page include this URL. (Twitter requires a valid domain name in the callback URL so it will not accept `http://localhost/` addresses.) After submitting the data, Twitter provides a consumer key and a consumer secret that must be entered into the plugin settings page. After completing these steps, Twitter now trusts the website.



Application Name:	Testing Elgg
Description:	An Elgg site
Application Website:	http://elgg.org/ Where's your application's home page, where users can go to download or use it?
Organization:	
Application Type:	Client [✓ Browser] Does your application run in a Web Browser or a Desktop Client?
Callback URL:	http://example.com/twitter_api/authorize Where should we return to after successfully authenticating?
Default Access type:	Read, Write, & Direct Messages [✓ Read & Write] Read-only What type of access does your application need? Note: @Anywhere applications require read & write access.

A user still has to tell Twitter the site has access to his account for the posting to work. This is done on the user's **Configure your tools** page in the **Settings** area. It includes a link to Twitter's authorization page.

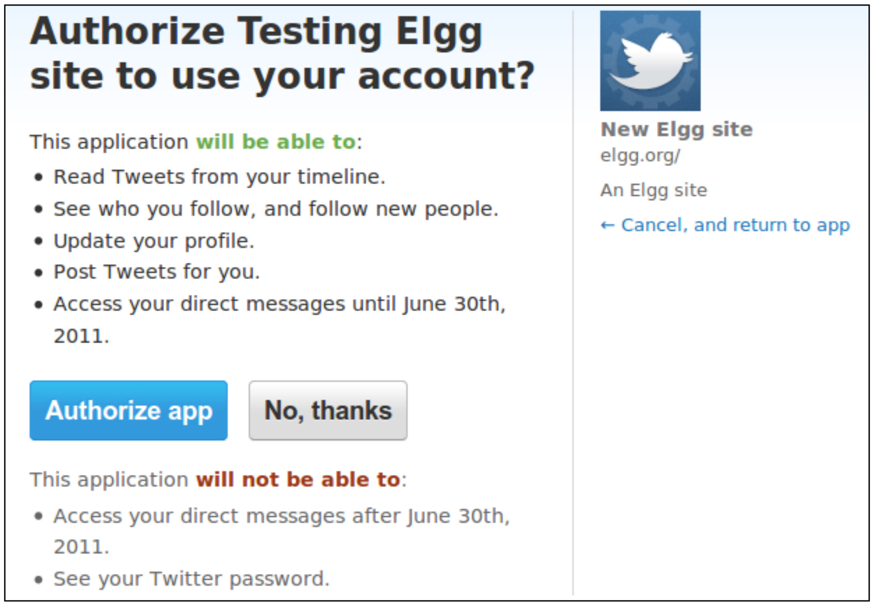


Twitter API

Link your Testing Elgg account with Twitter.

You must first authorize Testing Elgg to access your Twitter account.

Twitter explains what access the Elgg site will have to the user's account on the authorization page. The user can approve or deny this access.



Authorize Testing Elgg site to use your account?

This application **will be able to**:

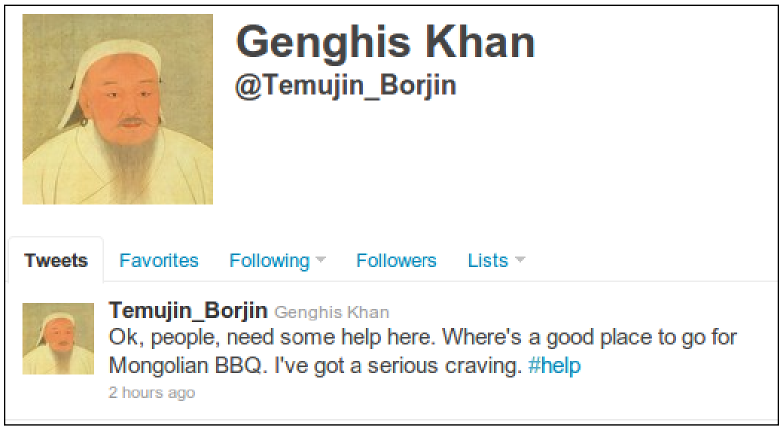
- Read Tweets from your timeline.
- See who you follow, and follow new people.
- Update your profile.
- Post Tweets for you.
- Access your direct messages until June 30th, 2011.

This application **will not be able to**:

- Access your direct messages after June 30th, 2011.
- See your Twitter password.

New Elgg site
elgg.org/
An Elgg site
← Cancel, and return to app

If the user approves access, the next wire post by the user is published on Twitter.


Genghis Khan
@Temujin_Borjin

Temujin_Borjin Genghis Khan
Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help
2 hours ago

Use cases

- **Answering questions**: On a corporate intranet, people post questions when looking for help: "How do I reserve the conference room on the third floor?" They can reach a large number of people through the wire. People get to control how they are notified (check the site occasionally, e-mail notifications, RSS feeds) rather than spending time reading a corporate-wide e-mail. Because the wire posts were published where everyone in the organization has access, the questions and answers can serve as a resource when people are looking for the same information in the future. Using the wire in this way does not replace other methods for seeking expertise in an organization, but can serve as a great complement.
- **Live tweeting an event**: Your students have been learning different propaganda techniques. The class is assigned to listen to a political speech on television and post to the wire as they detect the speaker using one of the techniques. The students can reply to each other on the wire during the speech. Later the timestamped wire posts can be compared by the class to a recording of the speech.

Customizations

Integration with Twitter was deemed important enough that Elgg distributes a plugin for this. This happens with other features that are commonly requested. They may start out as a community plugin, but later are integrated with the plugins bundled with Elgg.

- **Integration with Twitter**: Many of your users may want what they post on the wire to show up on Twitter. The `twitter_api` plugin distributed with Elgg does just that. A bonus is that people's tweets will contain a link to your site giving you some free advertising.
- **URL shortener**: If you expect your users to be posting links often, adding a URL shortener plugin is recommended.

Summary

Elgg has a powerful set of tools for sharing content. We covered the features provided by these tools so that can you try them out on your site. Beyond learning about the capabilities of the tools, we talked about design-related issues. Think like your users as you design your site. A great way to accomplish this is by developing use cases that describe how users might accomplish various tasks. From there you can develop lists of features and customizations needed to implement those use cases.

In the next chapter, we continue our in-depth exploration of the plugins distributed with Elgg by looking at the collaborative and communication plugins.

5 Communities, Collaboration, and Conversation

People naturally tend to form communities. They eat lunch with the same people again and again, get together with the same friends on the weekends, and talk with the same co-workers throughout the day. Social interactions on the Web are no different. Bloggers tend to collect into communities as they link to and comment on each other's posts. In discussion forums, there are often groups of participants who form cliques and have very personal interactions that extend beyond the purpose of the forum.

When running a social website, it is important to encourage and support the development of communities. Elgg does this through its **groups** tool, which helps organize people and content. Elgg also comes with tools for collaboratively editing content and for communicating with other users. As in the previous chapter, the discussion of each tool is divided into four sections: creating content, viewing content, sample use cases, and possible customizations.

The plugins covered in this chapter are as follows:

- Groups
- Pages
- Messages
- Message board

Groups

Groups are the primary hub of activity on most Elgg sites. The communities that form around the groups offer a great starting place for new members to find people with similar interests. The combination of the discussion forum and content-sharing tools creates a rich environment for community interactions. As you design your site, consider ways to feature the groups capability to potential and current users.

Features of the groups plugin include:

- Group profile
- Group activity
- Membership invitations and requests
- Membership list
- Discussion forum
- Content sharing
- Search for groups

Creating a group

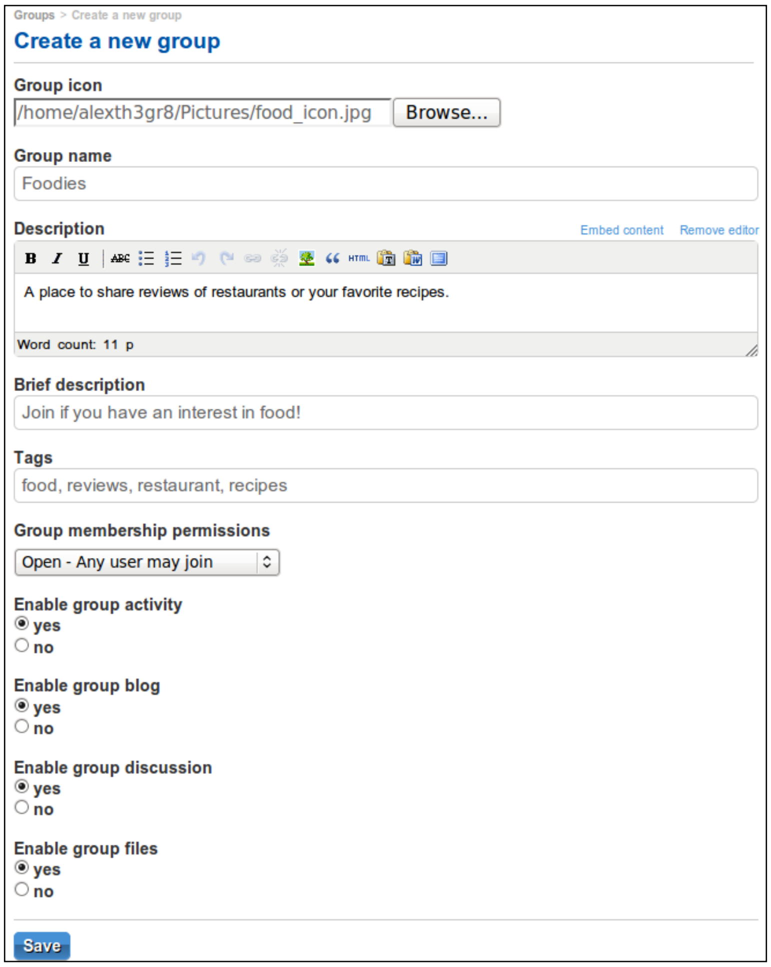
In Elgg, any member of a site can create a group. The button for group creation is found on the main group page, reached by selecting **Groups** from the site menu. An example group creation form is shown in the following screenshot. It includes:

- Group name
- Group icon
- Group profile fields: full description, brief description, and tags
- Membership (open or closed group)
- Enabling group tools (blog, bookmarks, discussions, files, pages)

The brief description field is used on the group listing page. It is supposed to catch people's attention as they scan the list of available groups. Search uses the full description field, so it is important to fill that out too.

Closed groups have restricted membership and their content pages are only available to members. Joining a closed group requires approval of the group owner, either by invitation or approved membership request.

Upon clicking the **Save** button, the group is created and the activity stream is updated to inform people about the new group. To update a group, the user who created the group (also known as the group owner) can click on the **Edit group** button on the group's profile page. The same group creation form is used, except that there is a **Delete** button at the bottom.



Group icon: /home/alexth3gr8/Pictures/food_icon.jpg

Group name: Foodies

Description: A place to share reviews of restaurants or your favorite recipes.

Word count: 11 p

Brief description: Join if you have an interest in food!

Tags: food, reviews, restaurant, recipes

Group membership permissions: Open - Any user may join

Enable group activity: [✓ yes] no

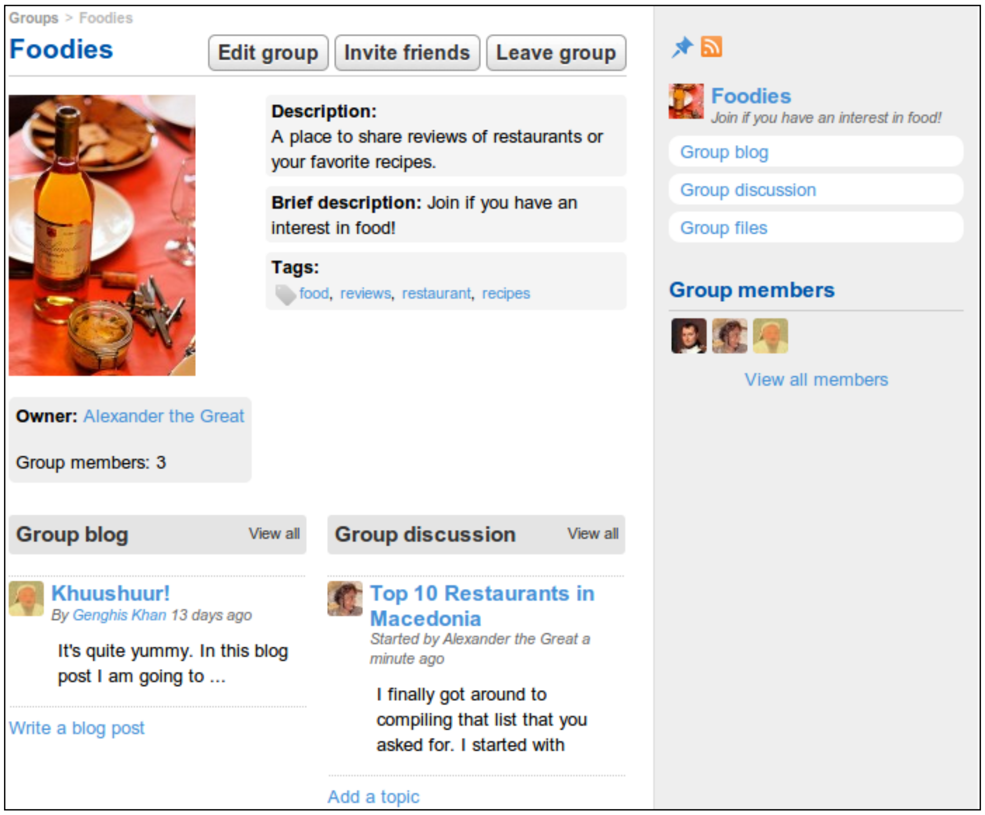
Enable group blog: [✓ yes] no

Enable group discussion: [✓ yes] no

Enable group files: [✓ yes] no

Group profile

The group profile page, as shown in the following screenshot, provides an overview of a group. The sidebar contains links for navigating through the group pages. It also includes a set of group member avatars with a link to a list of all the group's members.



Groups > Foodies

Foodies

Edit group | Invite friends | Leave group

Description: A place to share reviews of restaurants or your favorite recipes.

Brief description: Join if you have an interest in food!

Tags: food, reviews, restaurant, recipes

Owner: Alexander the Great

Group members: 3

Group blog — View all

Khuushuur!
By Genghis Khan 13 days ago

It's quite yummy. In this blog post I am going to ...

Write a blog post

Group discussion — View all

Top 10 Restaurants in Macedonia
Started by Alexander the Great a minute ago

I finally got around to compiling that list that you asked for. I started with

Add a topic

Foodies
Join if you have an interest in food!

Group blog
Group discussion
Group files

Group members

View all members

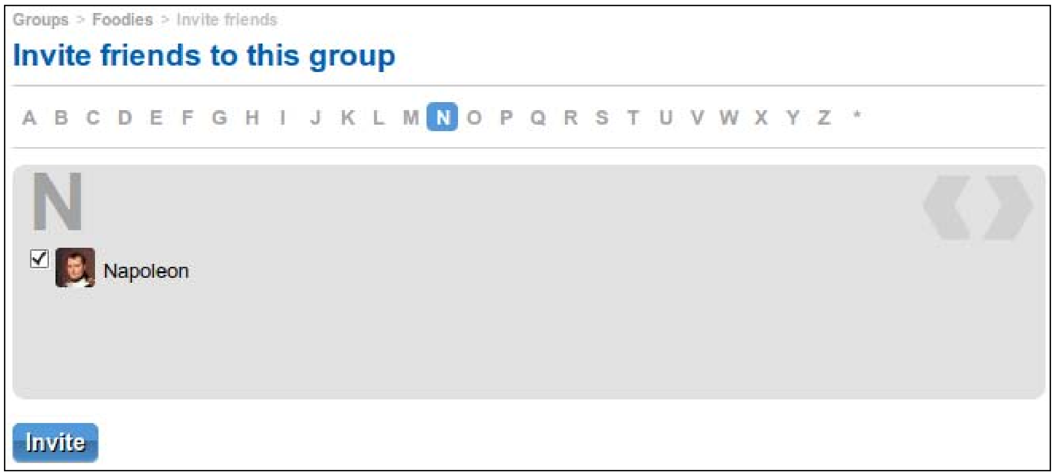
On the left side of the page are the profile fields and a set of widgets that display the latest group content. Unlike the widgets on a user's profile page, these widgets are not draggable.

Group RSS feed

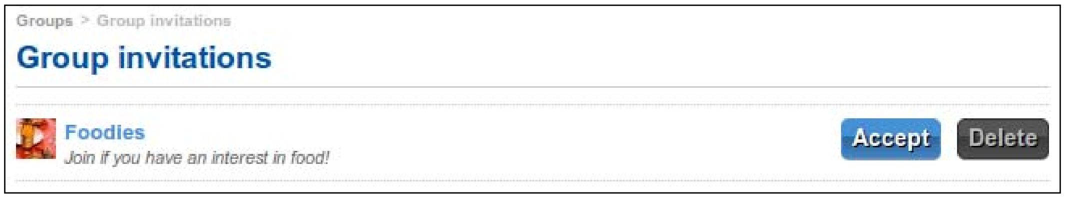
The group homepage has its own RSS feed and it is updated whenever a new discussion topic is created or when one of the other group tools is used. This provides a great option for those users who prefer to not receive e-mail notifications.

Membership

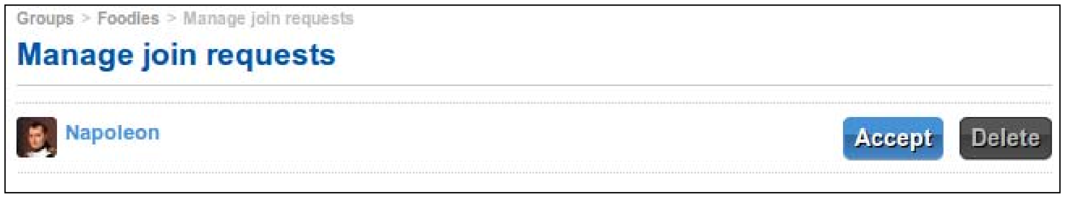
Only the group owner can invite new members. At the top of the group profile is a button that leads to the group invitation page. Group owners can select from among their friends as shown in the following screenshot. They then click on the **Invite** button to send out the e-mail invitations.


Groups > Foodies > Invite friends
Invite friends to this group
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z *
N
Napoleon
Invite

The e-mail invitation lets the receiver know who is inviting them to join the group and includes a link to that user's group invitations page. This page lists the invitations that the user has not responded to yet. The user can accept the invite or delete it.


Groups > Group invitations
Group invitations
Foodies
Join if you have an interest in food!
Accept
Delete

If a user wants to join a closed group, then there is a `Request membership` button where the `Join group` button is located for open groups. Clicking it sends a notification e-mail to the group owner. The owner manages pending requests for the group with an interface similar to the invitations page, as shown in the following screenshot:



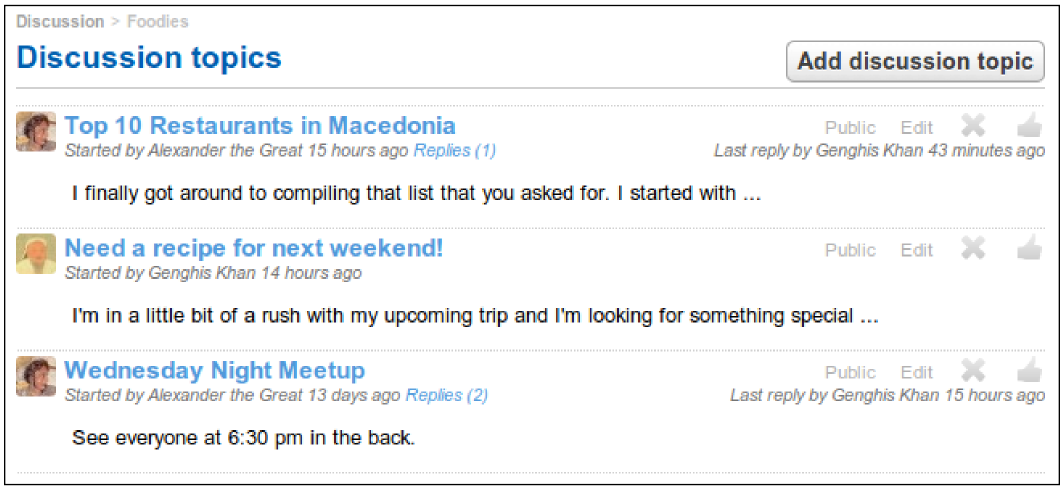
Groups > Foodies > Manage join requests

Manage join requests

Napoleon

Discussion forum

One of the features that make groups so attractive to users is that each group has its own forum. Group members can use the forum to ask questions, share news, or discuss whatever they are thinking about. The discussion forum can be reached through a sidebar menu item on the group profile page or the view all link in the discussion widget. Each topic summary lists the member who started it, when it was started, the number of replies, and when the last reply occurred.



Discussion > Foodies

Discussion topics

Top 10 Restaurants in Macedonia
Started by Alexander the Great 15 hours ago Replies (1)
Last reply by Genghis Khan 43 minutes ago

I finally got around to compiling that list that you asked for. I started with ...

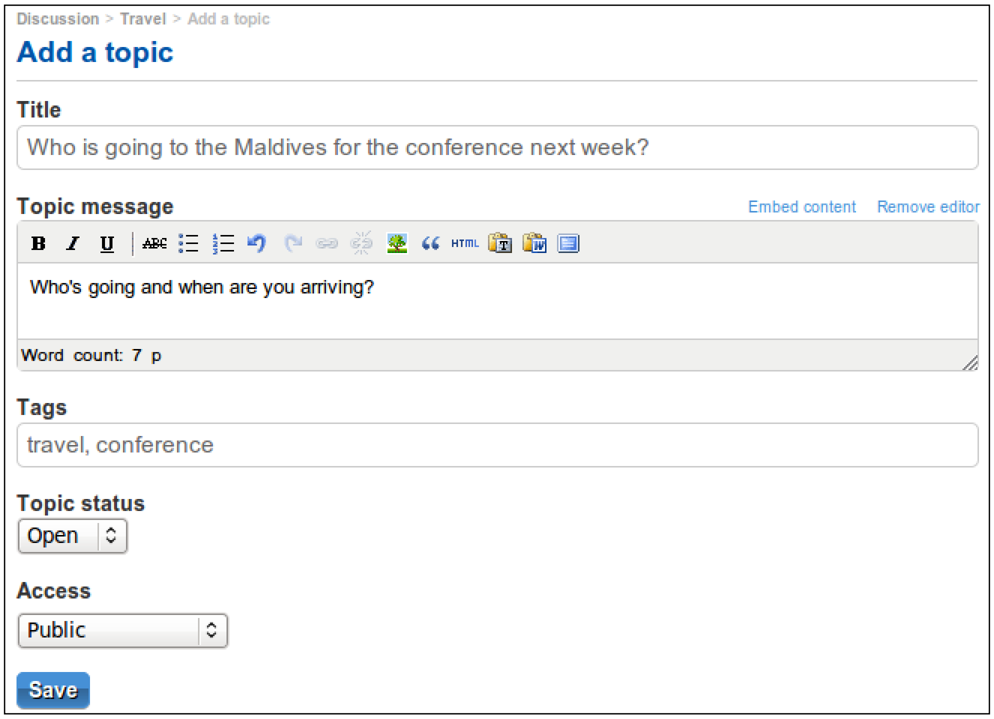
Need a recipe for next weekend!
Started by Genghis Khan 14 hours ago

I'm in a little bit of a rush with my upcoming trip and I'm looking for something special ...

Wednesday Night Meetup
Started by Alexander the Great 13 days ago Replies (2)
Last reply by Genghis Khan 15 hours ago

See everyone at 6:30 pm in the back.

Each topic has a title, body, tags, access level, and a topic status (open or closed topic). The WYSIWYG editor (discussed in *Chapter 4, Sharing Content*) and the image embedding capability are available when creating a topic. When a user submits a new topic, the topic is posted and notifications are sent to the members who registered for them.



Title: Who is going to the Maldives for the conference next week?

Topic message: Who's going and when are you arriving?

Word count: 7 p

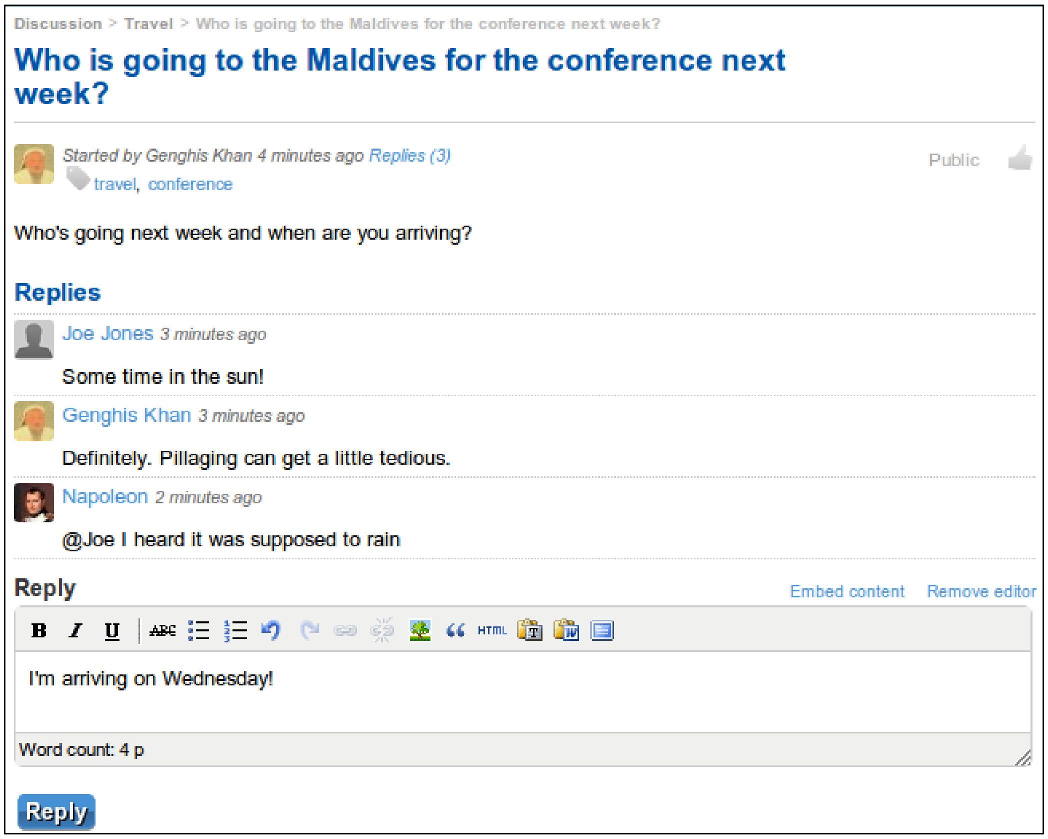
Tags: travel, conference

Topic status: Open

Access: Public

Changing the title of a topic or closing an open topic is done by clicking on the edit link on the topic. Each topic also has a delete link for removing it. The users with the permissions to use these links are the starter of the topic, the owner of the group, and any administrator.

Members of the group can respond to a posted topic by submitting comments. These comments are sent as notifications to those who registered for group notifications. The comments can be edited after they have been created so that those embarrassing spelling errors can be corrected. As with most pages on Elgg, users can subscribe to the RSS feed for that page. Each topic has a feed for its comments. There is also a feed for all new topics.


Discussion > Travel > Who is going to the Maldives for the conference next week?

Who is going to the Maldives for the conference next week?

Started by Genghis Khan 4 minutes ago Replies (3)

travel, conference

Who's going next week and when are you arriving?

Replies

Joe Jones 3 minutes ago

Some time in the sun!

Genghis Khan 3 minutes ago

Definitely. Pillaging can get a little tedious.

Napoleon 2 minutes ago

@Joe I heard it was supposed to rain

Reply
Embed content
Remove editor

I'm arriving on Wednesday!

Word count: 4 p

Group tools

Besides the discussion forum, the group owner can turn on four other tools: blogs, bookmarks, files, and pages. The only difference between using these tools within a group is the notifications. When a group member writes a blog post, members who registered for group notifications are told about it. That person's friends are not notified. Instead, the notifications are sent to the group members who have enabled group notifications.

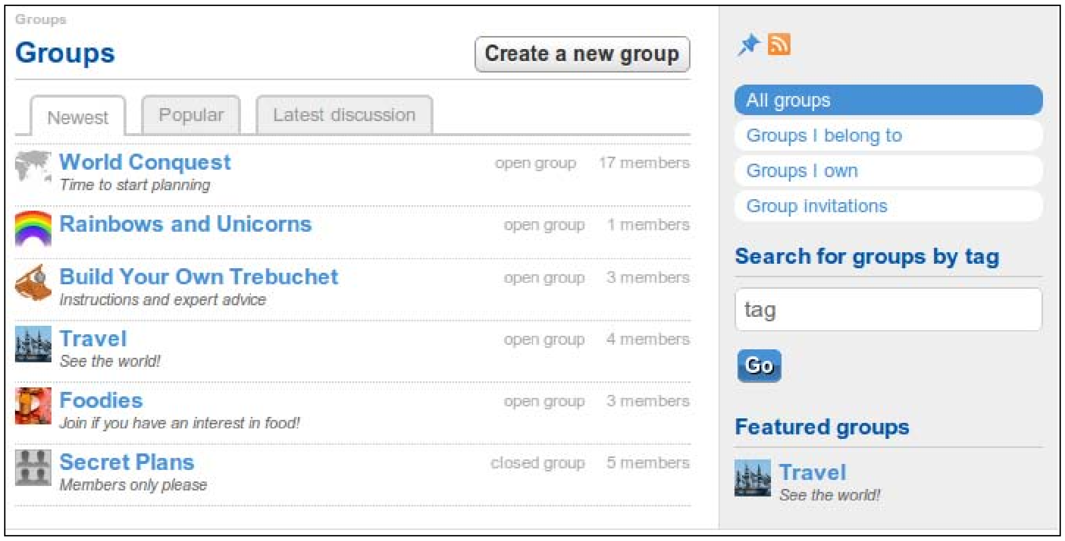
A question that comes up frequently regarding the group tools is when to use the discussion forum, blog, and pages within a group. They overlap in functionality and this can be confusing – especially for new users. A good metaphor to distinguish these tools is the following:

- The group blog is a lot like a conference where a single speaker is presenting at a time. When the speaker is done, people can interact through questions.
- A discussion forum topic is like a conference call or meeting where individual people take turns leading the conversation.
- The group pages are like people standing around a whiteboard working on a design or a plan. As different people draw on the board, they can add to, change, or erase what the others put up before them.

Elgg-based sites are not limited to the content tools that are distributed with Elgg. Additional tools can be added to groups through plugins. We cover a few possibilities in *Chapter 6*, *Finding and Using Plugins*.

Finding groups

As more and more groups are created on your site, users have more difficulty scanning all of them to find the ones that interest them. There are three tools for this on the main group page. First, the newest and most popular groups are listed on separate tabs as shown in the following screenshot. The other tab lists the latest discussion across all groups. Using these tabs enables users to quickly look for interesting discussions and groups.



World Conquest — Time to start planning — open group — 17 members
Rainbows and Unicorns — open group — 1 members
Build Your Own Trebuchet — Instructions and expert advice — open group — 3 members
Travel — See the world! — open group — 4 members
Foodies — Join if you have an interest in food! — open group — 3 members
Secret Plans — Members only please — closed group — 5 members

Second, there is a search box in the sidebar. This searches over the tags that the group owners have added to the group profiles. Third, the site administrator can select particular groups to be featured using the **Make featured** link that is available in the newest and popular tabs. The featured groups appear in the sidebar. This is a good way to advertise the more interesting groups to your users.

Use cases

- **Community of practice -** There are employees spread throughout the organization with a similar skill. They need a place to ask questions and share best practices. This encourages the flow of information across the organization. A group provides the tools needed for this.
- **Group project** - Your students have been divided into teams for a research project. Each team has a group so that they can discuss the assignment, share information, and collaborate on writing the report. At the end of the project, the group is an artifact that can be used to evaluate the teams' work.
- **Premium content** - Your site offers free membership, but you derive income by offering premium content for a fee. Access to this content is controlled through closed groups. Joining a closed group requires payment of the fee.

Customizations

The following are some possible customizations for the groups tool that can be implemented through plugins. Many of them are available in the Elgg community plugin repository, which we cover in the next chapter.

- **Group categories** - As the number of groups grows, it can become difficult for new members to find the groups that they want to join. Organizing the groups into categories is very helpful.
- **Group widgets** - The group profile page has widget-like areas to display the latest blogs or files, but group owners have no control over the layout or how many items are displayed. A user profile-like widget system gives the group owner more control.

- **Group administrators** - As groups grow in size, more than just the group owner may be required to administer the group.
- **Group moderation** - You may want to limit the creation of groups to prevent the formation of unofficial groups or duplicate groups.
- **Rename to communities** - If you think the name 'communities' better captures what is happening with the groups tool, then you can rename it through a simple plugin.

Pages

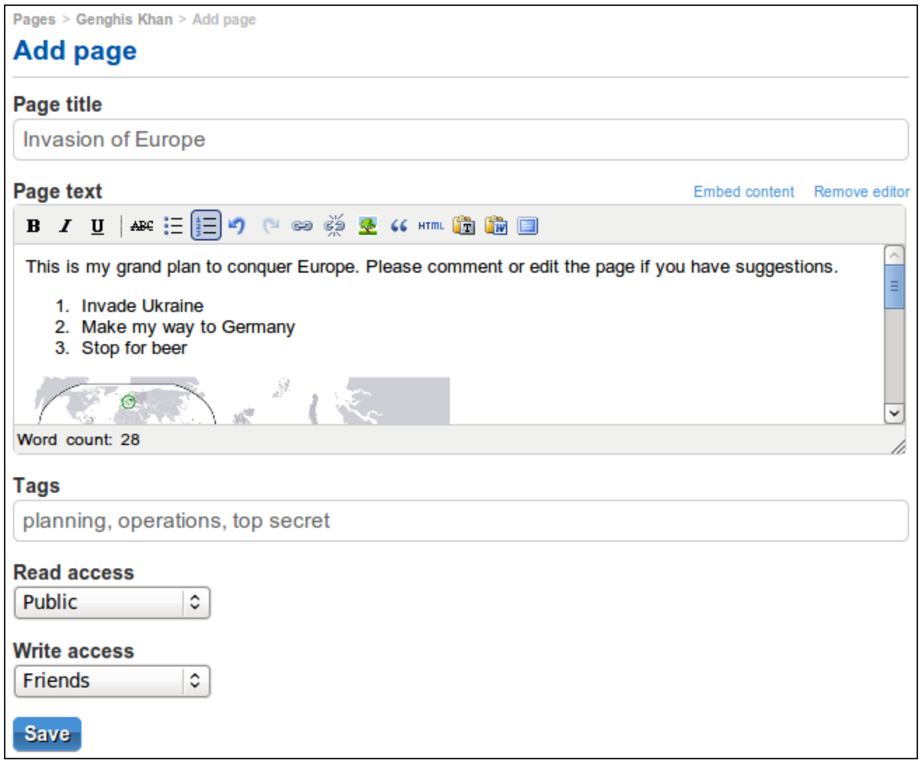
The pages tool enables users to put together text and images to create simple web pages. It works like a wiki in that multiple people can edit the pages, but it does not require that users learn wiki syntax. It provides capabilities similar to a collaborative word processor.

The features provided by the pages plugin and the other plugins distributed with Elgg include the following:

- WYSIWYG Editor
- Embedding photos
- Collaborative editing
- Tags
- Read and write access control
- Nested pages
- Commenting
- Liking

Creating pages

When users create a new page, they are presented with a familiar editing interface. In fact, there is only one difference between the interfaces for writing a blog and writing a page: pages include both a read and a write access level control. The write access can be set to only group members if the page belongs to a group or a user could set the write access so that only friends can edit it. An example of creating a page is shown in the following screenshot:



Page title: Invasion of Europe

Page text:

This is my grand plan to conquer Europe. Please comment or edit the page if you have suggestions.

1. Invade Ukraine
2. Make my way to Germany
3. Stop for beer

Word count: 28

Tags: planning, operations, top secret

Read access: Public

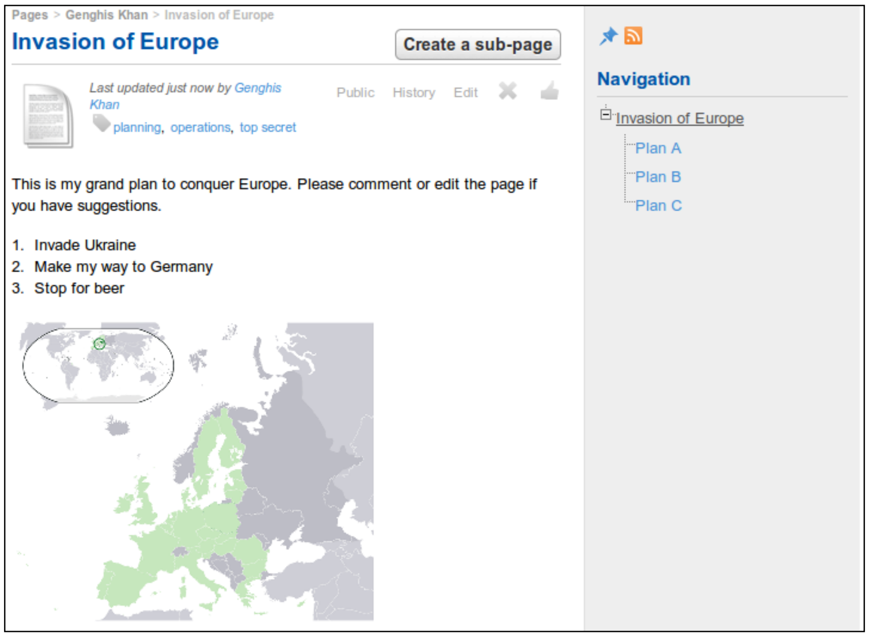
Write access: Friends

When a user saves a new page, the activity stream is updated and e-mails are sent to those who registered for notifications. When a page is updated, the activity stream gets a new entry, but no notifications are sent.

Pages can be organized in a hierarchical fashion by creating sub-pages. A sub-page is created when viewing a page by clicking on the **Create a sub-page** button at the top of the content area. There is no limit to the number of levels in the page hierarchy that can be created.

Viewing

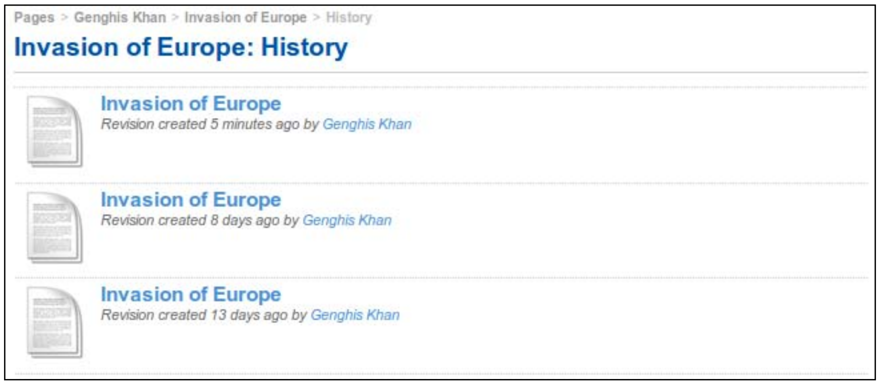
Clicking on **Pages** in the site menu takes users to a list of the latest pages. There are separate tabs for the user's and friends' pages. Only top-level pages are displayed in these lists. Clicking on a page's title leads to a web page that displays the content and provides a navigation box in the sidebar for any sub-pages.


Pages > Genghis Khan > Invasion of Europe
Invasion of Europe
Create a sub-page
Last updated just now by Genghis Khan
planning, operations, top secret
Public
History
Edit
Navigation
Invasion of Europe
Plan A
Plan B
Plan C

This is my grand plan to conquer Europe. Please comment or edit the page if you have suggestions.

1. Invade Ukraine
2. Make my way to Germany
3. Stop for beer

Each page has an RSS feed that includes the latest version of the page. To see the previous versions of a page, users can click on the **History** link that is next to the edit link. The history page lists the revisions ordered by date and each revision can be viewed.


Pages > Genghis Khan > Invasion of Europe > History
Invasion of Europe: History

Invasion of Europe
Revision created 5 minutes ago by Genghis Khan

Invasion of Europe
Revision created 8 days ago by Genghis Khan

Invasion of Europe
Revision created 13 days ago by Genghis Khan

Use cases

- **Help documentation** - You want to add some documentation to your site. The pages tool provides a convenient way to write and organize this documentation.
- **Group-based FAQ** - A popular use for the pages tool is collecting frequently asked questions within a group. It saves everyone time if this resource is available to both long time and recent members.

Customizations

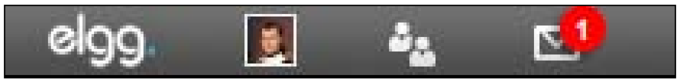
- **Restore a revision** - With collaborative editing, there will come a time when the users want to roll back to a previous revision. This is done by copying the content from that revision and saving it over the current version. A built-in capability to do that with a single click of a button would be useful.

Messages

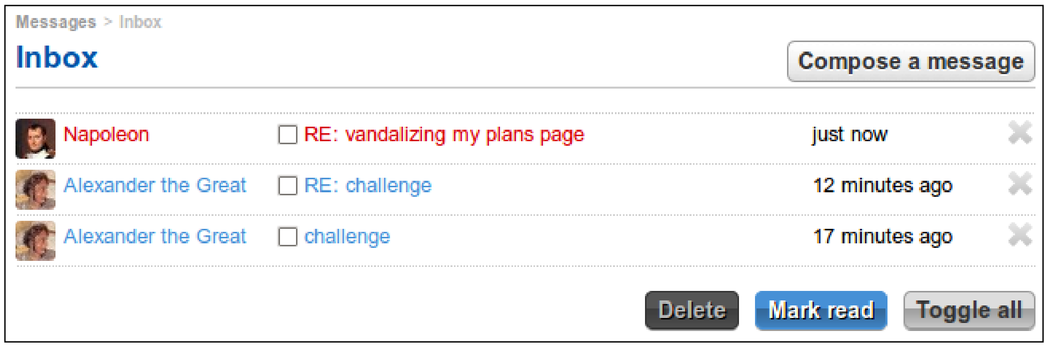
There are times when users want to communicate privately. The messages tool provides an e-mail-like interface for exchanging messages without revealing e-mail addresses.

Inbox

Next to the friends icon in the top bar is a white envelope icon. When a user has new private messages, a red circle appears with the number of unread messages. Clicking the envelope icon takes a user to the user's inbox.



In the inbox, unread messages are highlighted in red. Each message has a **Delete** button and there are controls below the message list for deleting multiple messages or marking all the displayed messages as read.

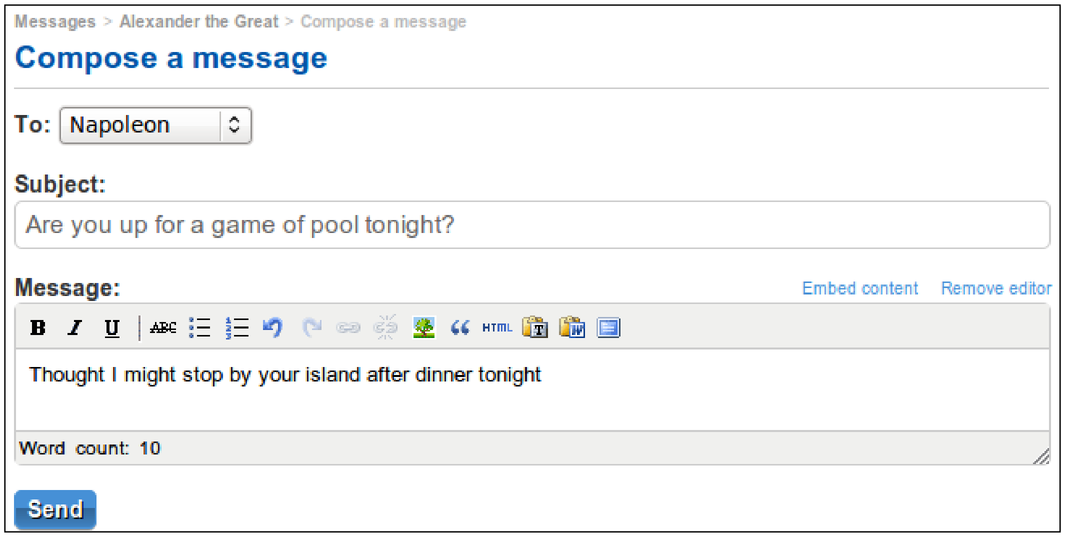

Messages > Inbox

Inbox

Napoleon	RE: vandalizing my plans page	just now	
Alexander the Great	RE: challenge	12 minutes ago	
Alexander the Great	challenge	17 minutes ago	

Composing

To write a private message, a user clicks on the **Compose a message** button. The drop-down menu for selecting the recipient is populated with that user's friends. After selecting the recipient, the user fills out the subject and body and clicks on **Send**. The message is stored in the user's sent messages folder and a notification is sent to the recipient.


Messages > Alexander the Great > Compose a message

Compose a message

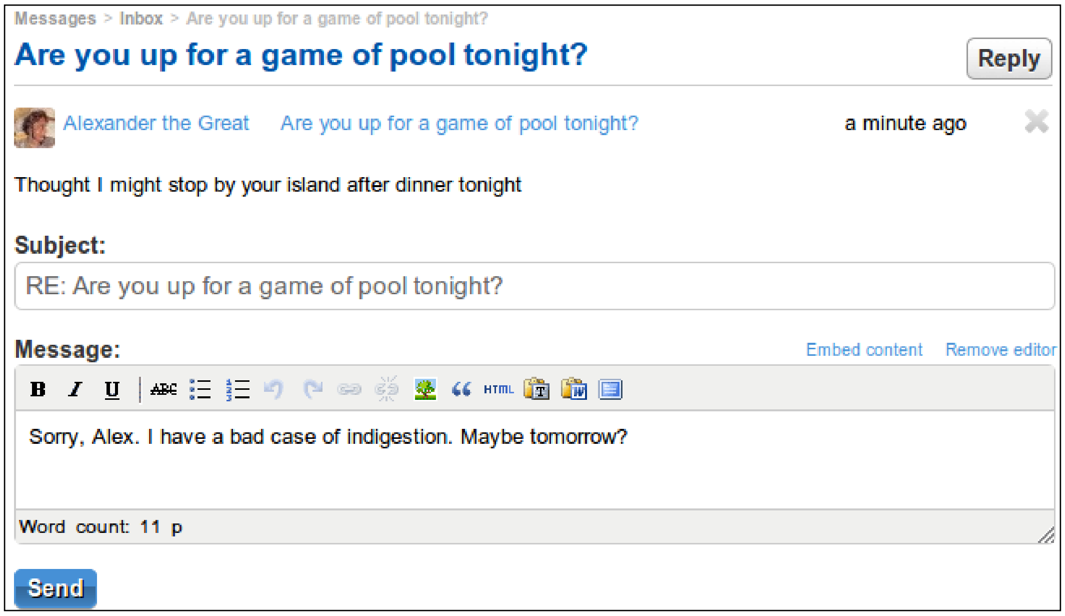
To: Napoleon

Subject: Are you up for a game of pool tonight?

Message: Thought I might stop by your island after dinner tonight

Word count: 10

To send a message to someone who is not a friend, a user can select the **Send a message** option in the hover menu on the other person's avatar. To reply to a message, a user clicks on the **Reply** button above the received message. A textbox appears with the subject already filled in as shown in the following screenshot:


Messages > Inbox > Are you up for a game of pool tonight?
Are you up for a game of pool tonight?
Reply
Alexander the Great
Are you up for a game of pool tonight?
a minute ago

Thought I might stop by your island after dinner tonight

Subject:

RE: Are you up for a game of pool tonight?

Message:

Embed content
Remove editor

Sorry, Alex. I have a bad case of indigestion. Maybe tomorrow?

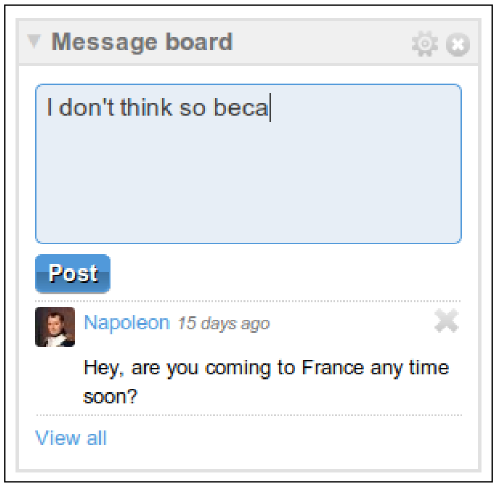
Word count: 11 p

Customizations

- **Limit to friends** - By default, Elgg allows anyone to send a message to anyone else. Users may not like this and spammers can take advantage of this. By limiting messaging to friends-only, you can handle both concerns.
- **Send email to groups** - The messages tool only supports sending a message to a single recipient. Being able to send to a list of users provides a more email-like functionality.

Message board

The message board adds a public conversation capability to Elgg similar to Facebook's Wall. The primary interface to the message board is a widget that users can put on their profile page. The following screenshot shows what this widget looks like:



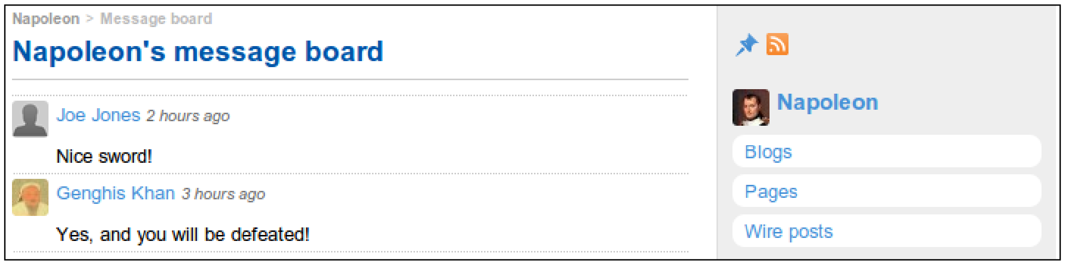
There are three links of interest on the message board widget:

1. The **View all** link leads to the message board of the widget's owner.
2. The **reply** link takes a user to the message board of the poster.
3. The **history** link is used to view posts between the poster and the user who is viewing the widget.

The distinction between these links can be confusing to first-time users, and it usually requires some experience to understand them.

When someone posts on a message board, a notification is sent to the message board's owner. The notification includes a link to the poster's message board page. It is usually best to post a reply on the other person's message board so that they get a notification.

The full message board page is shown in the following screenshot. It is reached by clicking on the widget's view all link or by following the link in a notification.



Customizations

- **Wall-to-wall** - The history page can be confusing because it does not show the conversation between the poster and widget owner, but the viewer and the poster. There is a plugin on the Elgg community site called wall-to-wall that makes the history link display the conversation between the poster and the person whose profile page you are on.
- **Guest book** - The message board could be renamed as a guest book. A guest book does not need to support conversations; the confusing reply and history links could be removed.

Summary

Community, collaboration, and conversation – three important aspects needed to make your social site…well, social. The groups tool is critical for encouraging collaboration and community development. The other tools covered provide complementary social capabilities. This finishes our exploration of Elgg and the plugins bundled with it.

In the next chapter, we cover how to find, install, and evaluate plugins in the Elgg community plugin repository.

6 Finding and Using Plugins

Elgg is distributed with over 30 plugins. The previous three chapters gave an overview of the features that they provide. It is possible to build a social networking site using only those plugins, but it is likely that you will want to customize your site with additional plugins. This chapter covers all you need to know to find, evaluate, install, and configure plugins. The topics will include the following:

- Administering plugins on your site
- Finding plugins on the Elgg community site
- Installing plugins
- Finding and installing themes
- Previewing three popular plugins available from the Elgg plugin repository

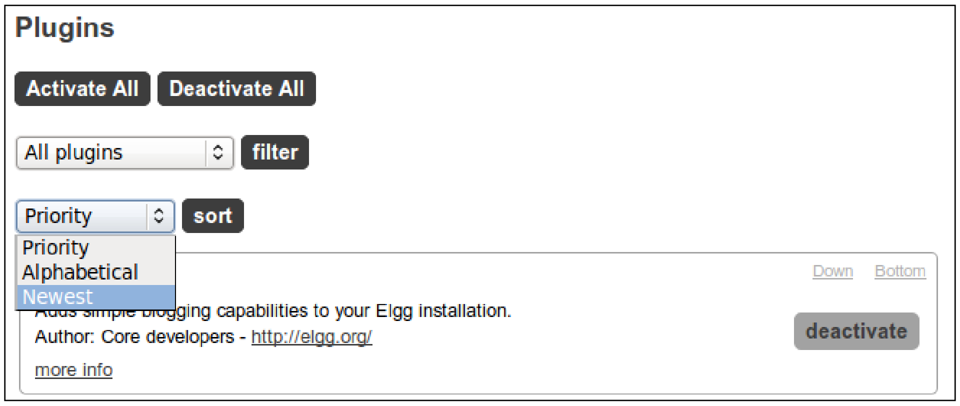
Plugin administration

In *Chapter 3, A Tour of Your First Elgg Site*, we gave a quick introduction to the plugin administration page by describing how to activate a plugin. Because Elgg is so plugin-focused, it needs a good tool for managing these plugins. This section provides detailed information on how to use the plugin administration page.

Filtering and sorting

One of the first things you may have noticed about the plugin page is that there are a lot of plugins bundled with Elgg, which results in a lot of scrolling to find any individual plugin. To simplify this, Elgg allows an administrator to filter the plugins. The filter has options for viewing only the plugins that are inactive. This makes it easy to locate a plugin that was recently uploaded to a server but not yet enabled. Plugins can be filtered by whether they were bundled with Elgg or were downloaded separately. Finally, plugins also have categories such as Admin, Multimedia, or Communication that can be used for filtering.

Besides filtering, Elgg also supports sorting plugins as demonstrated in the following screenshot. The list of plugins can be sorted by three criteria: priority, alphabetical, and newest. As we discuss next in this section, the order in which Elgg loads plugins matters. Priority sorting displays this order so that it can be adjusted. The newest option sorts plugins by the date they were first uploaded to the server. If you think a recent plugin addition has been causing problems on your site, then you can use this sort choice to find the possible culprit.

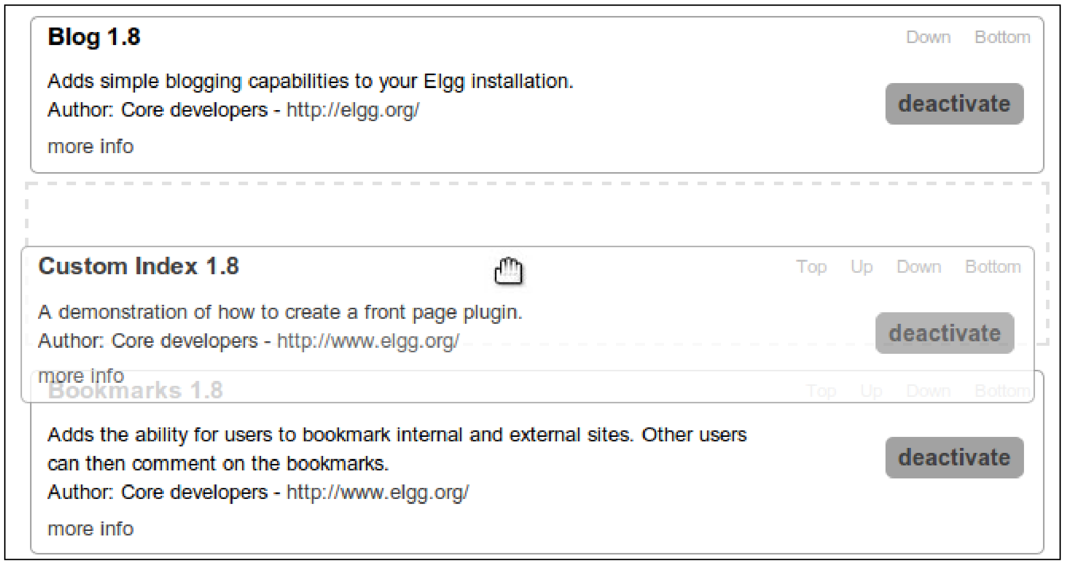


Adds simple blogging capabilities to your Elgg installation.
Author: Core developers - http://elgg.org/

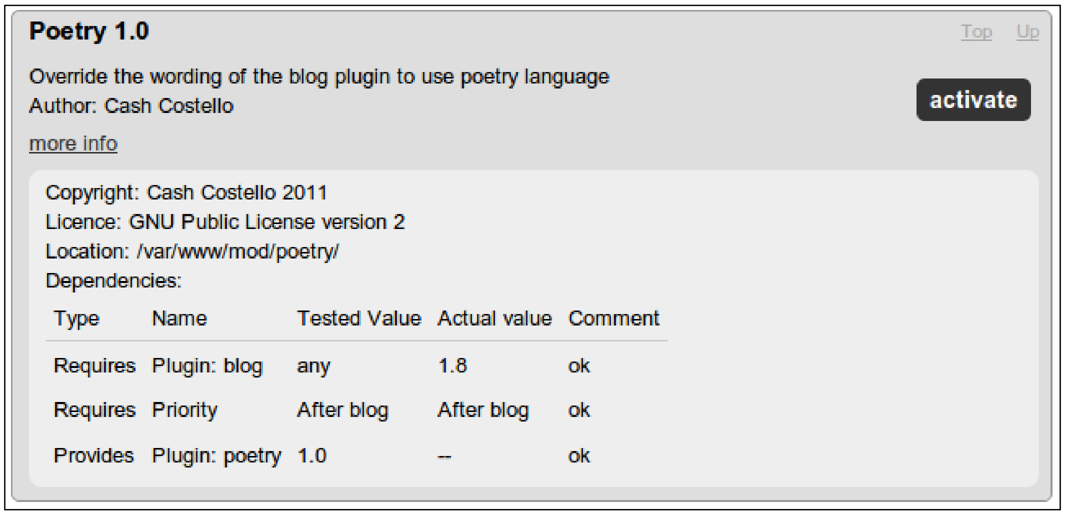
Plugin dependencies

Plugins not only add new capabilities, but also modify existing functionality. Consider a theme plugin as an example of modifying Elgg. It might modify Elgg's built-in theme by changing the color scheme, but leave the fonts the same. A plugin can also modify another plugin works. An example of this is a plugin that changes the titles and wording of the blog plugin to use *poetry* instead of the word *blog*. For this override to work, the plugin must be loaded after the blog plugin.

By default, newer plugins are loaded last. If the loading order must be changed, then there are **Up** and **Down** links on each plugin for moving a plugin a single spot at a time. Plugins can also be dragged when you move the mouse over the top of a plugin box as shown in the following screenshot. This makes it easy to move a plugin several slots in the priority order. As a precaution, reordering is only allowed when the plugins have been sorted by priority.


Blog 1.8
Down Bottom
Adds simple blogging capabilities to your Elgg installation.
Author: Core developers - http://elgg.org/
deactivate
more info
Custom Index 1.8
Top Up Down Bottom
A demonstration of how to create a front page plugin.
Author: Core developers - http://www.elgg.org/
deactivate
more info
Bookmarks 1.8
Top Up Down Bottom
Adds the ability for users to bookmark internal and external sites. Other users can then comment on the bookmarks.
Author: Core developers - http://www.elgg.org/
deactivate
more info

For most plugins, order does not matter. All of the plugins bundled with Elgg can be loaded in any order. If a plugin modifies how another plugin works, then it needs to be loaded after that plugin. Fortunately, the plugin system tells you when order matters as long as the plugin author included this information in the plugin description. The **more info** link toggles the display of all the dependencies and other information about the plugin. As shown in the following screenshot, the poetry plugin says that it must be loaded after the blog plugin:

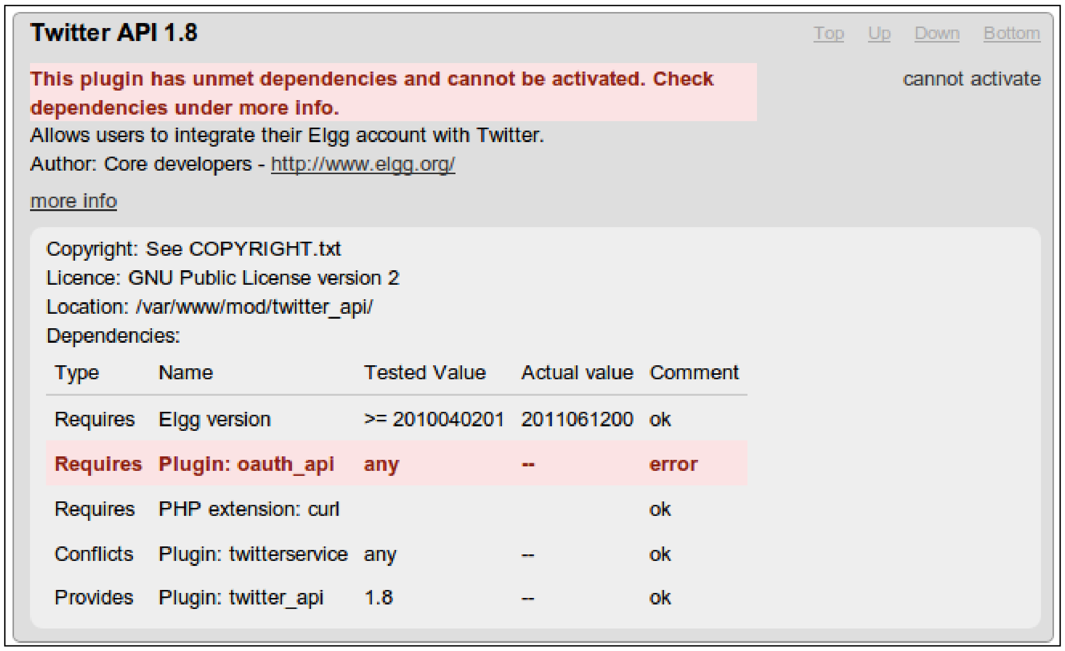

Poetry 1.0

Override the wording of the blog plugin to use poetry language
Author: Cash Costello

Copyright: Cash Costello 2011
Licence: GNU Public License version 2
Location: /var/www/mod/poetry/
Dependencies:

Type	Name	Tested Value	Actual value	Comment
Requires	Plugin: blog	any	1.8	ok
Requires	Priority	After blog	After blog	ok
Provides	Plugin: poetry	1.0	--	ok

Plugins can depend on more than the order in which they are loaded. Some plugins only work with certain versions of Elgg. For example, Elgg 1.8 added a new menu system. A plugin that uses this menu system will not work with Elgg 1.7. A plugin can also require that another plugin is enabled before it can be activated. In *Chapter 4, Sharing Content*, we described how the Twitter API plugin requires the `OAuth` plugin. When the `OAuth` plugin is not enabled, the plugin system highlights the failed dependency as shown in the following screenshot and does not allow the Twitter API plugin to be enabled:



Twitter API 1.8

This plugin has unmet dependencies and cannot be activated. Check dependencies under more info.

Allows users to integrate their Elgg account with Twitter.

Author: Core developers - http://www.elgg.org/

Copyright: See COPYRIGHT.txt

Licence: GNU Public License version 2

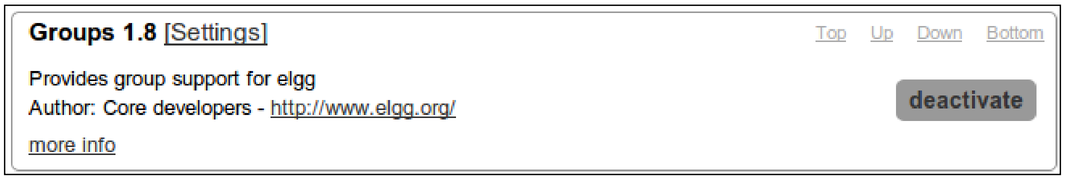
Location: /var/www/mod/twitter_api/

Dependencies:

Type	Name	Tested Value	Actual value	Comment
Requires	Elgg version	>= 2010040201	2011061200	ok
Requires	**Plugin: oauth_api**	**any**	**--**	**error**
Requires	PHP extension: curl			ok
Conflicts	Plugin: twitterservice	any	--	ok
Provides	Plugin: twitter_api	1.8	--	ok

Plugin settings

The plugins that have administrative settings also have links next to their names that lead to their respective settings pages.

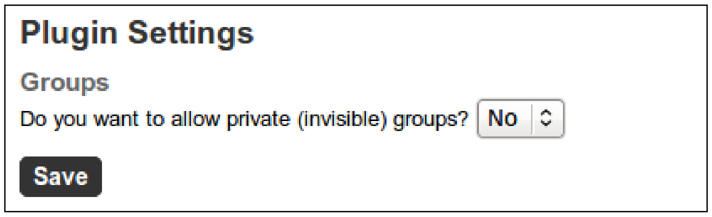

Groups 1.8 [Settings]

Provides group support for elgg

Author: Core developers - http://www.elgg.org/

more info

An alternate way to reach a plugin's settings page is through the sidebar menu. Under `Configure | Settings` are pages for configuring the site (basic and advanced settings) and for plugins. A plugin settings page ranges from a single drop-down option as shown in the following screenshot to extensive options offered on multiple tabs:


Plugin Settings

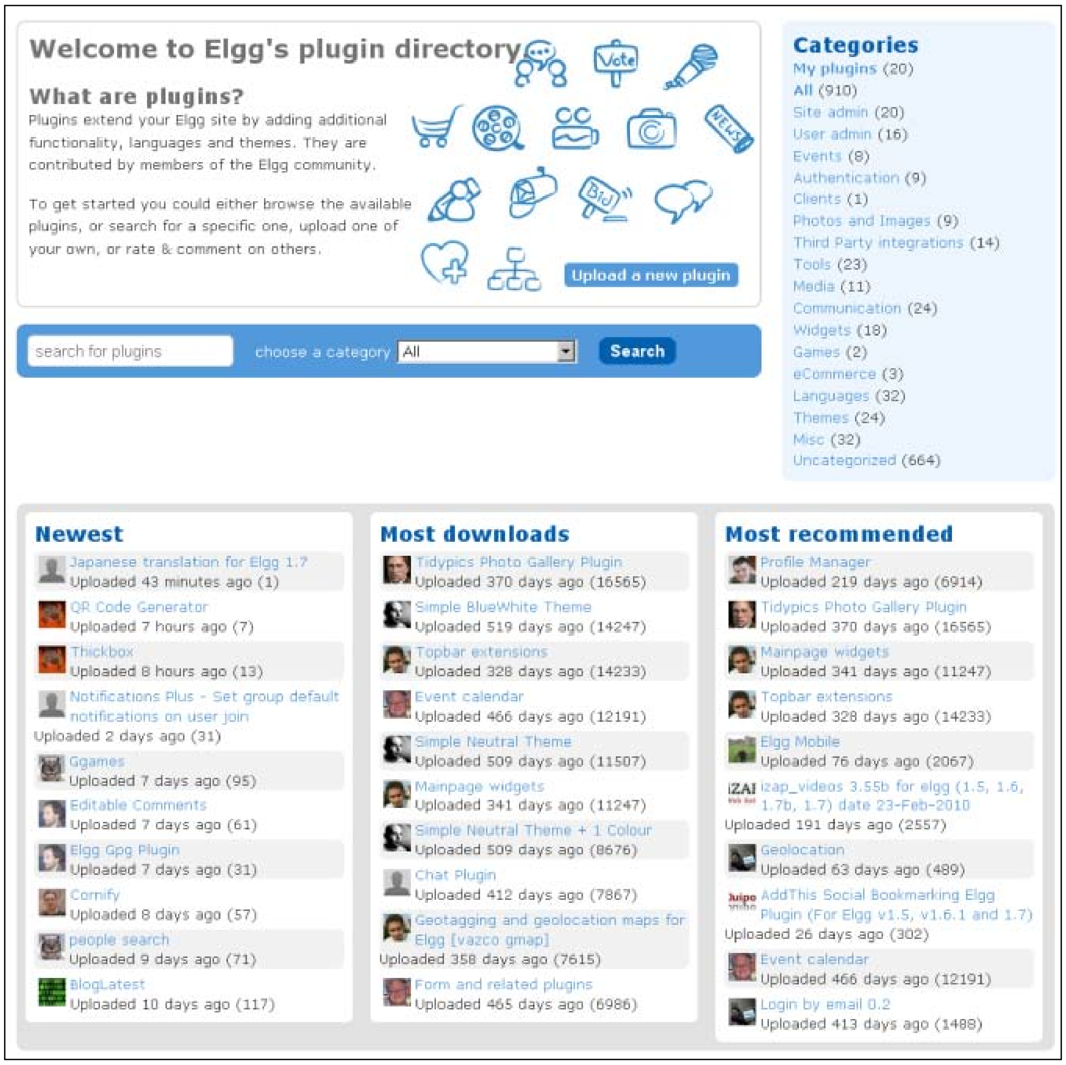
Groups

Do you want to allow private (invisible) groups? No

Finding plugins

After experimenting with Elgg for the last three chapters, you probably have a list of what you would like to add to or change about the site. The next step is to find plugins that do what you want. The first place to look is the Elgg community plugin repository (`http://community.elgg.org/pg/plugins/all/`), as shown in the following screenshot. Hundreds of people have uploaded over a thousand plugins to the repository. The quality of the plugins in the repository varies greatly. Many were written by professional developers who work with Elgg as a full-time job. Others were coded by software developers who like to dabble in plugin development in their free time. Still others were created by people with no prior development experience, but who have figured out how to customize some aspect of Elgg and want to share it with others.

In this section, we cover how to find plugins in the community plugin repository and how to evaluate their quality before downloading them. We conclude the section by discussing your options if you do not find plugins that satisfy all of your requirements.



Welcome to Elgg's plugin directory

What are plugins?

Plugins extend your Elgg site by adding additional functionality, languages and themes. They are contributed by members of the Elgg community.

To get started you could either browse the available plugins, or search for a specific one, upload one of your own, or rate & comment on others.

Categories
My plugins (20)
All (910)
Site admin (20)
User admin (16)
Events (8)
Authentication (9)
Clients (1)
Photos and Images (9)
Third Party integrations (14)
Tools (23)
Media (11)
Communication (24)
Widgets (18)
Games (2)
eCommerce (3)
Languages (32)
Themes (24)
Misc (32)
Uncategorized (664)

Newest
Japanese translation for Elgg 1.7
Uploaded 43 minutes ago (1)
QR Code Generator
Uploaded 7 hours ago (7)
Thickbox
Uploaded 8 hours ago (13)
Notifications Plus - Set group default notifications on user join
Uploaded 2 days ago (31)
Ggames
Uploaded 7 days ago (95)
Editable Comments
Uploaded 7 days ago (61)
Elgg Gpg Plugin
Uploaded 7 days ago (31)
Cornify
Uploaded 8 days ago (57)
people search
Uploaded 9 days ago (71)
BlogLatest
Uploaded 10 days ago (117)

Most downloads
Tidypics Photo Gallery Plugin
Uploaded 370 days ago (16565)
Simple BlueWhite Theme
Uploaded 519 days ago (14247)
Topbar extensions
Uploaded 328 days ago (14233)
Event calendar
Uploaded 466 days ago (12191)
Simple Neutral Theme
Uploaded 509 days ago (11507)
Mainpage widgets
Uploaded 341 days ago (11247)
Simple Neutral Theme + 1 Colour
Uploaded 509 days ago (8676)
Chat Plugin
Uploaded 412 days ago (7867)
Geotagging and geolocation maps for Elgg [vazco gmap]
Uploaded 358 days ago (7615)
Form and related plugins
Uploaded 465 days ago (6986)

Most recommended
Profile Manager
Uploaded 219 days ago (6914)
Tidypics Photo Gallery Plugin
Uploaded 370 days ago (16565)
Mainpage widgets
Uploaded 341 days ago (11247)
Topbar extensions
Uploaded 328 days ago (14233)
Elgg Mobile
Uploaded 76 days ago (2067)
izap_videos 3.55b for elgg (1.5, 1.6, 1.7b, 1.7) date 23-Feb-2010
Uploaded 191 days ago (2557)
Geolocation
Uploaded 63 days ago (489)
AddThis Social Bookmarking Elgg Plugin (For Elgg v1.5, v1.6.1 and 1.7)
Uploaded 26 days ago (302)
Event calendar
Uploaded 466 days ago (12191)
Login by email 0.2
Uploaded 413 days ago (1488)

Browsing the repository

Looking through what is available in the repository will likely give you new ideas for your site. There are plugins that add significant capabilities to your site, such as a photo gallery or a chat client. There are also many that tweak the way that Elgg works or add a simple feature.

The front page of the plugin repository lists the newest, most downloaded, and most recommended plugins. The last two are a great place to start because the Elgg community has already indicated that those are great plugins. The plugins are also organized by categories. If you are looking for a certain type of plugin that fits one of the categories, then you can scan through the category listing looking for a match.

While not required to download plugins, it is a good idea to register for an account on the Elgg community site. Having an account enables you to ask questions of the plugin developers or interact with other Elgg users in the forums.

Searching

The repository supports searching over the title and descriptions of plugins. Searches can be limited to a particular category (show me all plugins in the category widgets that include the word *video*). The search engine used by the repository is very rudimentary and you may have more success using Google.

Evaluating before downloading

It is a great feeling when you try out a plugin that adds terrific features to your Elgg site. All it takes is a few minutes of your time and the generosity of the developer that shared the plugin. The opposite is also possible: you are in a panic when a newly installed plugin crashes your site. Here are some tips to give you more of the first feeling and hopefully, none of the second.

Look at the plugin overview

Is the plugin description well-written? Does it adequately explain the purpose of the plugin? How many times has the plugin been downloaded? Does it have many recommendations?

Read the comments

Are people complaining about the plugin or praising the plugin author? Does the author answer people's questions? Are there bugs that have not been fixed?

Check the history

Has the plugin been updated often? While it is possible that the first release did not have any bugs, more often plugin authors need to make a few releases to work out some of the problems. Has the plugin been updated recently? Recent activity indicates the author is maintaining the plugin and possibly adding new features.

Find developers that you trust

The best way to avoid poorly written plugins is to find the best developers and use their plugins. Many of them upload plugins that they build for clients. You can list all of a developer's plugins by using the hover menu on a user's profile avatar on the community site.

Custom plugins

If you did not find what you were looking for by browsing and searching, then the remaining option is to create a custom plugin. If you have development skills or a strong interest in learning, then you can build it yourself. *Chapters 7* and *8* provide a tutorial-based introduction to plugin development. If you are not inclined to create your own, then you can contract with a developer to write plugins for you. To find a developer, you can start by checking the list of development companies that advertise on the Elgg site (`http://elgg.org/services.php`) or by contacting the developers of plugins that you like. Be aware that not all of them do custom plugin development, but instead focus on creating entire websites based on Elgg.

Installing plugins

It is very tempting to download every interesting plugin and try them all at once. Don't. If something does go wrong, then it is very difficult to figure out which plugin is causing the problem. The best way to try out new plugins is to methodically add one at a time, exploring the functionality of the plugin before moving on to the next one. As you do this, you may want to take notes about user interface details that you would like to change or features that you would like to add or modify. These notes will help you later to prioritize and estimate the amount of effort involved in getting your site up and running.

Test server

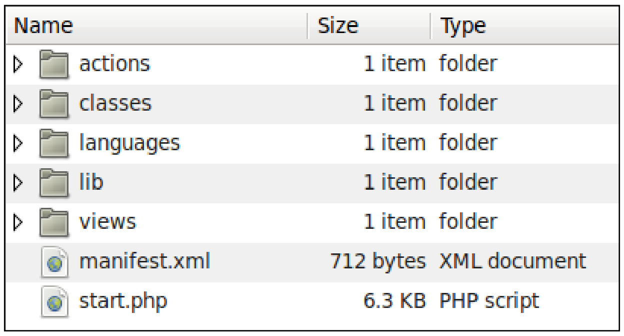
Testing new plugins is not conducive to running a stable site. It is always best to test plugins on a test server. You should never try new plugins on a production site. Make a mistake with your production site and your visitors could be greeted by an empty white screen. Always test any changes on a test site before deploying them to a production site. A test site could be a laptop with a few fake test users or another server with a clone of the production database.

Production site

A production site is one with real users who expect the site to be stable and functional.

Copying the code

After you have found and downloaded a plugin that you want to try, the next step is to put the plugin code into the `mod` directory of Elgg. When you downloaded the plugin, it was likely archived as a ZIP file. Extract the files and check that the plugin's base directory has files and directories that look something like the following screenshot:



Name	Size	Type
actions	1 item	folder
classes	1 item	folder
languages	1 item	folder
lib	1 item	folder
views	1 item	folder
manifest.xml	712 bytes	XML document
start.php	6.3 KB	PHP script

Then copy that plugin directory into the `mod` directory of Elgg. In the `mod` directory, you should have directories such as blog, bookmarks, and groups and they should all have this same structure.

Activating and configuring

After copying the code, navigate to the plugin page on your Elgg site. Check that there are not any errors due to unmet dependencies and then activate the plugin. The plugin should change color from inactive grey to active white.

Invalid plugin?

If Elgg says the plugin is invalid, then there are three possible causes, as follows:

1. The web server may not be able to read the files so the permissions need to be changed.
2. Sometimes when the plugin is extracted from the `zip` archive, there are two levels of directories before you get the files shown on the preceding page. If that is the case, then copy the second directory in Elgg's mod directory.
3. The last possibility is that it really isn't a plugin. If you do not see a file named `start.php`, then you can be sure that it is not a plugin.

After the plugin is enabled, configure any settings that it has and then start testing.

Troubleshooting

If a new plugin causes a significant problem with your site, then disable it. If you cannot reach the plugin administration page, then create a file in the `mod` directory called `disabled`. It does not need to have any content. Elgg checks if that file exists and if it does, it does not load any of the plugins. You can then disable the offending plugin and delete the `disabled` file.

If you experience problems with a plugin, then you have a variety of resources available to you. Each plugin's page in the community plugin repository has an area for comments so that you can communicate with the author and other users of the plugin. There are also group forums on the community site that can be used for getting help or advice. Furthermore, do not forget to check the Elgg wiki (`http://docs.elgg.org`) that has plenty of great troubleshooting information. There are links to these resources at the bottom of every Elgg administration page.

Themes

If you have scanned the Elgg directory structure, then you have not found a directory labeled `themes`. This is because in Elgg, a theme is a plugin. Themes live in the same directory as plugins. They have the structure of plugins. They are activated like plugins. This is unlike many of the web applications that you may have used in the past.

Since a theme is a plugin, you may be wondering where the theme plugin distributed with Elgg is. There isn't one. The default theme is built into Elgg. You do not turn off the default theme and replace it with another theme. Instead, a theme plugin modifies the default theme. To use an analogy, theming with Elgg is like remodeling a house rather than building a new one. Painting the rooms of the house can create a very different look. You can also rip out some walls if the floor plan does not match what you want. The amount of work depends on how different the current house is from your dream home. The same is true for theming. You can achieve a wide variety of looks by replacing the default CSS of Elgg. If the default HTML does not support your site design, then you can also selectively replace it through a theme plugin.

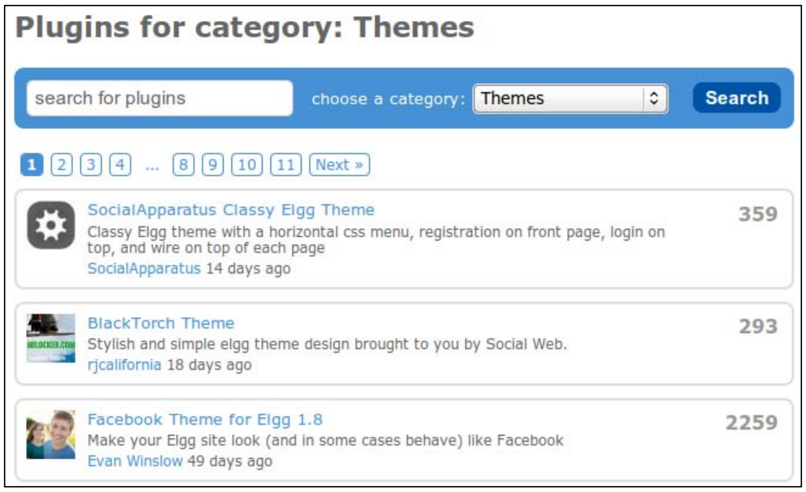
All themes override the default CSS, which determines visual aspects such as the color scheme and fonts. The CSS also combines with the HTML to determine the layout. Most themes include graphics that match the color scheme determined by the CSS. Many also change a small amount of HTML and include additional JavaScript libraries to add user interface elements such as slideshows, animations, or fancy forms.

Finding and installing themes

You have four options when looking for a theme: free themes, commercial themes, custom themes, or building your own. This section provides an overview of the first three options. *Chapter 9*, *Theming Elgg,* discusses creating your own theme.

Free themes

The Elgg community site is the first place to look for themes. In the plugin area of the site, you can select the theme category to see a listing of what is available.



Plugins for category: Themes

SocialApparatus Classy Elgg Theme 359
Classy Elgg theme with a horizontal css menu, registration on front page, login on top, and wire on top of each page
SocialApparatus 14 days ago

BlackTorch Theme 293
Stylish and simple elgg theme design brought to you by Social Web.
rjcalifornia 18 days ago

Facebook Theme for Elgg 1.8 2259
Make your Elgg site look (and in some cases behave) like Facebook
Evan Winslow 49 days ago

The themes are uploaded primarily by theme designers who are showcasing their work or by people sharing a theme they created for their site. Most themes' pages include screenshots so that you can preview the theme before downloading it. As with any of the plugins on the site, the quality varies from theme to theme. Read through the comments on a theme to evaluate other users' experiences with it. The number of times a theme has been downloaded also lets you know which ones are the better themes.

Commercial themes

A commercial theme is one that is available for purchase by the public. It is offered at a price that is a small percentage of the cost to produce it. The designer is depending on selling the theme to many users in order to make a profit. The commercial theming community for Elgg is still small as compared to the WordPress community or other popular web applications like Drupal or Magento. Googling *Elgg themes* will turn up a handful of sites that specialize in selling Elgg themes. When evaluating a commercial theme designer, consider the following:

- Try a free theme by the same theme designer. Does it work well or does it break your site? Are there useful comments in the code to help with customizing it?
- Ask about support. Some designers will sell you the theme as-is while others will provide limited support as you work with the theme.
- Ask about the designer's policies. Can you get your money back if the theme does not work well for you? Are there any restrictions on how you can modify it, such as having to leave a link to their site in the footer? Do you get free upgrades?

Custom themes

While free and commercial themes cost little to nothing, you know that there will be other Elgg-based sites using the same theme. You may also need to spend some time tweaking the theme to fit your needs. Contracting with a designer/developer to create a custom theme guarantees a unique look for your site, but at a significantly higher cost than a commercial theme.

When looking for a front-end developer or web designer to build a theme, you are not limited to those who advertise experience with Elgg. You are looking for a developer with strong visual design skills and some basic knowledge of PHP. Ideally, the developer has had experience creating themes for similar web applications. A developer who has not worked with Elgg before would need to learn how to override and extend views. For a theme that deviates significantly from the standard HTML markup of Elgg, the developer would also need to learn the basics of writing Elgg plugins. The time spent learning Elgg will generally factor into the cost of theme.

Installing themes

Themes install just like plugins, as follows:

1. Copy the theme directory into `/mod`.
2. Activate the plugin (and disable your previous theme if you were not using Elgg's default theme).
3. Optionally set any theme options.

Themes are typically placed last in the plugin list so that they can override the CSS of other plugins.

After the theme has been installed, look around your site to check for problems.

> If there are sections of your site that are not modified by the theme (that is, the colors are wrong), then it is likely that you have a plugin that includes its own CSS that the theme does not override. You may be able to get assistance from the developer of the theme to fix this. Otherwise, you must make the necessary modifications to the theme. This will not happen with a custom theme as you already provided the developer with a list of plugins to support.

When testing, be sure to use multiple browsers. It is especially important to test with **Internet Explorer** (**IE**), the bane of web designers everywhere. It is not uncommon for a theme to work perfectly in Firefox, Safari, and Chrome, but fall apart in IE.

As you try out different browsers and different pages with your new theme, keep a list of what is not working or what you would like to change. You may decide the list is too long and look for a different theme or the list may become your to-do list in your first foray into theme customization.

Did the theme break all the JavaScript on your site?

If the drop-down menus stop working and the status messages stop fading away, then this means that there is an error in the theme's JavaScript code or the theme included a version of jQuery that is incompatible with your version of Elgg. Check with the theme developer to determine what versions of Elgg the theme has been tested on. The technical support group forum on the Elgg community site is also a good resource for getting troubleshooting advice.

Major community plugins

There are hundreds of plugins available in the community plugin repository (`http://community.elgg.org/`). There is no way to review them all in this book, so we will highlight three of them. Each of these plugins is very popular within the Elgg community and is maintained by skilled developer(s). They are among the best demonstrations of what can be created with a plugin in Elgg.

Tidypics

Tidypics is a full-featured photo sharing tool that is actively developed.

Plugin profile

Author: Cash Costello with contributions by several other developers.

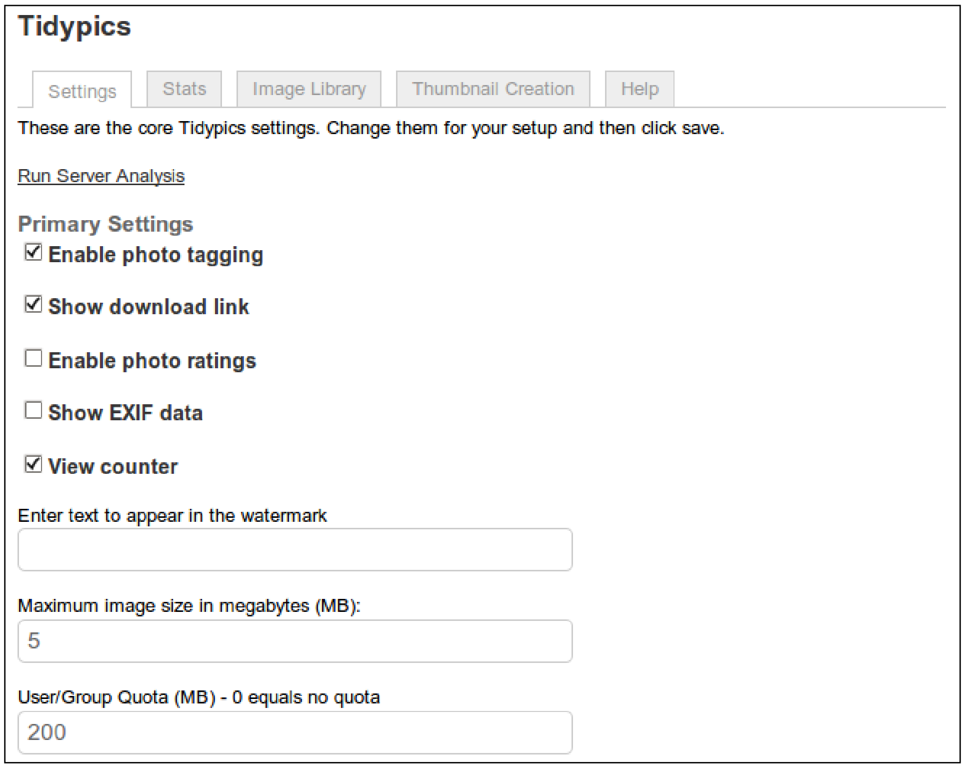
Link: `http://community.elgg.org/pg/plugins/tidypics/read/385077`

Features include:

- Photo albums
- Photo tagging
- Slideshow
- Watermarking
- Album sorting
- Bulk image uploading
- User quotas
- Commenting
- Groups integration

Administration

Many of the major plugins available in the community repository have a large number of configuration options. This enables customization of the plugin without writing any code. Tidypics is no exception to this. About half of its settings are shown in the following screenshot. Many of these settings have been requested by people just like you over the last three years. There are settings for the thumbnail sizes to better match your theme and a per-user disk quota so that your hard drive is not filled up with photos. A view counter for each image can be enabled and a watermark can be configured. Glance at the settings to see the full range of what is offered.



These are the core Tidypics settings. Change them for your setup and then click save.

Run Server Analysis

Primary Settings

☑ Enable photo tagging

☑ Show download link

☐ Enable photo ratings

☐ Show EXIF data

☑ View counter

Enter text to appear in the watermark

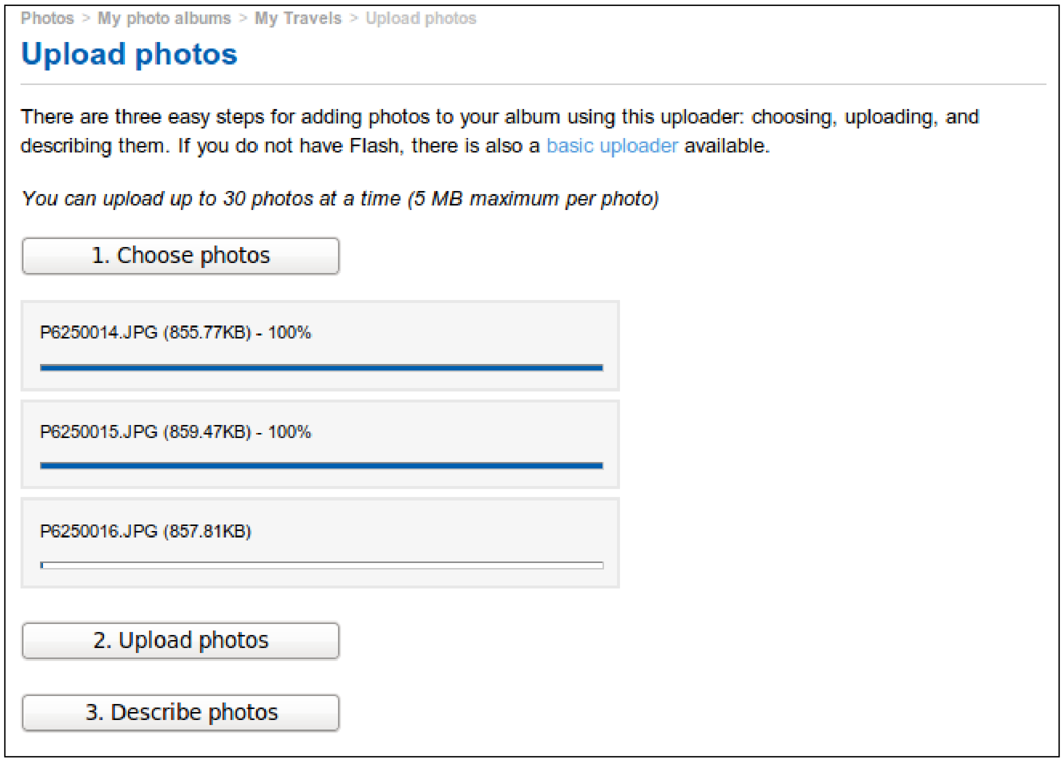
Maximum image size in megabytes (MB): 5

User/Group Quota (MB) - 0 equals no quota: 200

In addition to the **Settings** tab, Tidypics has four other tabs with administrative information. Because Tidypics needs to resize images for thumbnails or add watermarks, it needs access to an image processing library. There are instructions on the **Image Library** tab for determining what image library is best for a site. There is also a **Help** tab with troubleshooting instructions for common issues.

Uploading photos

Tidypics organizes photos into albums. Each album has its own page with description, comments, tags, and categories. An album can belong to an individual user or to a group. Photos are uploaded and added to the albums. There are two methods for uploading photos: a basic upload form with slots for up to 10 images and a Flash-based uploader that can handle up to 30 images at a time. The Flash uploader displays the progress of each image transfer, as shown in the following screenshot:


Photos > My photo albums > My Travels > Upload photos
Upload photos

There are three easy steps for adding photos to your album using this uploader: choosing, uploading, and describing them. If you do not have Flash, there is also a basic uploader available.

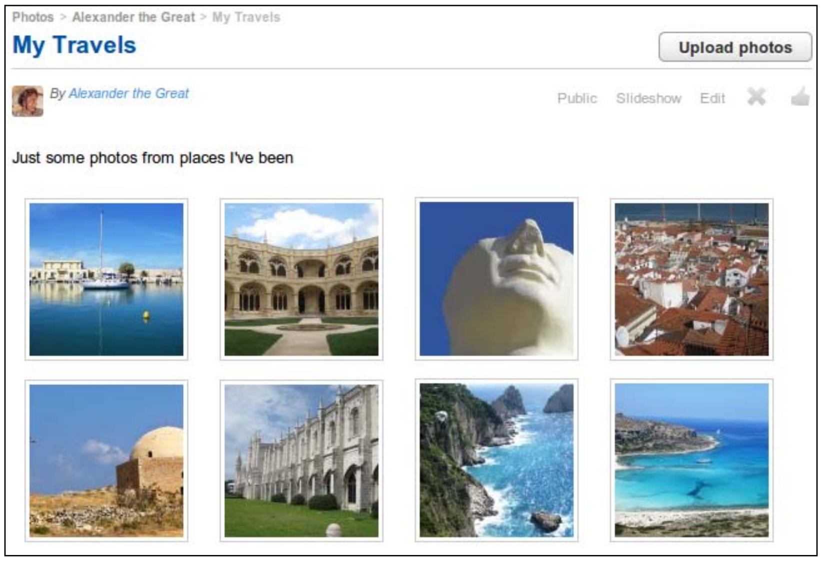
You can upload up to 30 photos at a time (5 MB maximum per photo)

1. Choose photos
P6250014.JPG (855.77KB) - 100%
P6250015.JPG (859.47KB) - 100%
P6250016.JPG (857.81KB)
2. Upload photos
3. Describe photos

Viewing photos

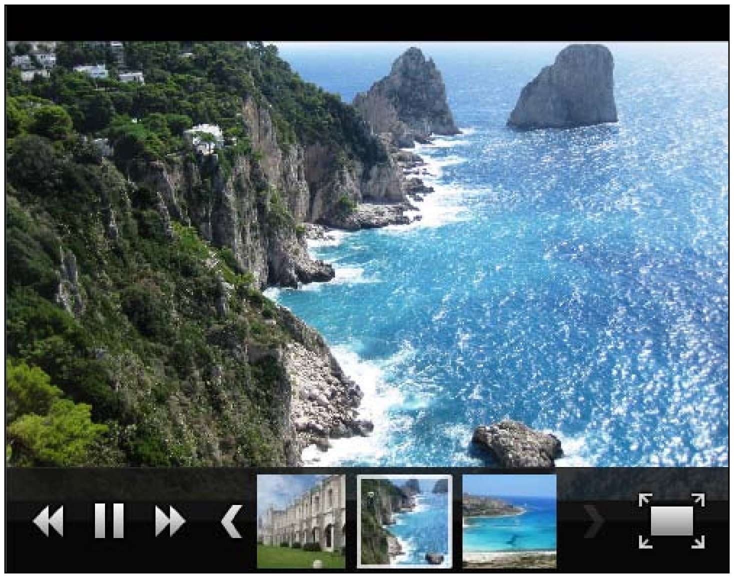
The owner of an album selects a photo to serve as its cover photo. This cover photo is used in the activity stream update, the album widgets, and any page that lists albums. Besides showing up in albums, photos also appear on collation pages. There are pages that display the most viewed photos, most commented, and most recently uploaded. These are all good browsing pages to facilitate discovery of new content by users.

The following screenshot shows an album's web page. The most recently added photos are displayed as a gallery. Clicking on an image leads to a larger view of the image. Comments can be left on individual images or on the album as a whole.



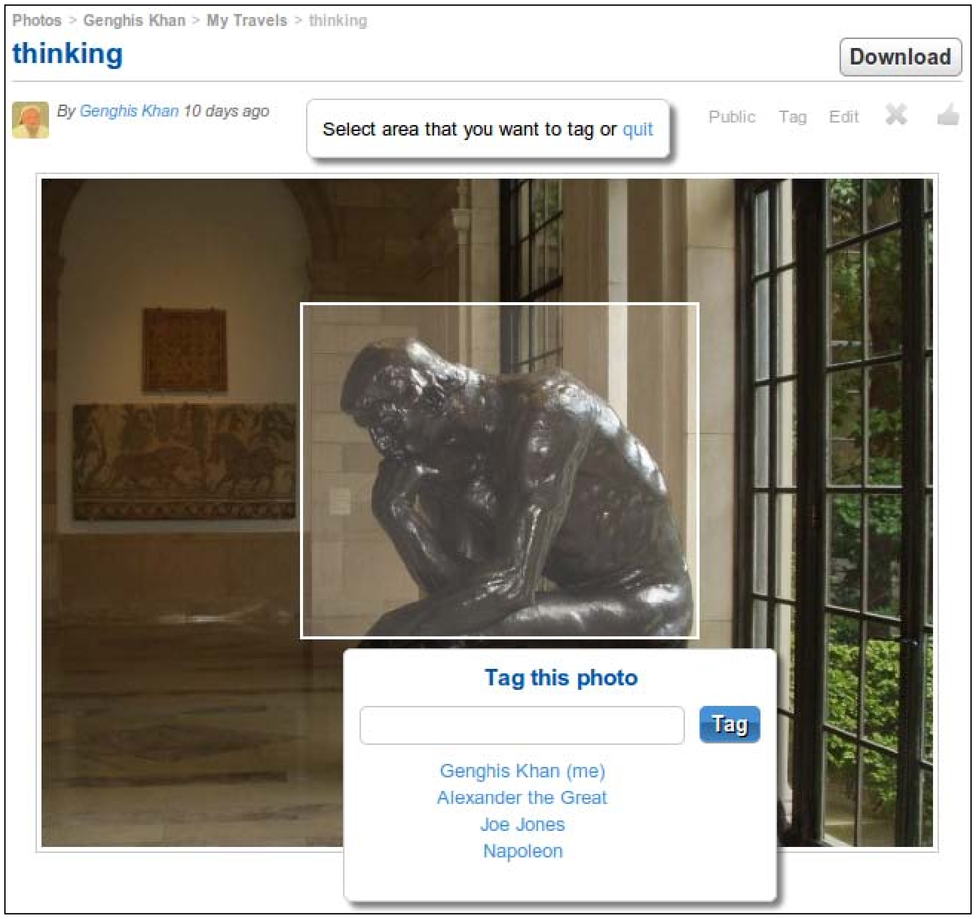
Just some photos from places I've been

In addition to viewing the photos on their web pages, each album has a slideshow link in its menu. The slideshow, shown in the following screenshot, provides a more dynamic way to view large numbers of images:



Tagging photos

A capability familiar to users of Facebook and Flickr is photo-tagging. With Tidypics, users can select portions of an image to identify a friend in the image. The people being tagged are notified of this. There is also a web page for displaying all the photos that a user has been tagged in.


Photos > Genghis Khan > My Travels > thinking
thinking
Download
By Genghis Khan 10 days ago
Select area that you want to tag or quit
Public
Tag
Edit
Tag this photo
Tag
Genghis Khan (me)
Alexander the Great
Joe Jones
Napoleon

Event calendar

This plugin extends Elgg to provide an event calendar. It offers a wide range of configuration options so that administrators can customize it for their sites.

Plugin profile

Author: Kevin Jardine of Radagast Solutions

Link: `http://community.elgg.org/pg/plugins/project/384926`

Features include:

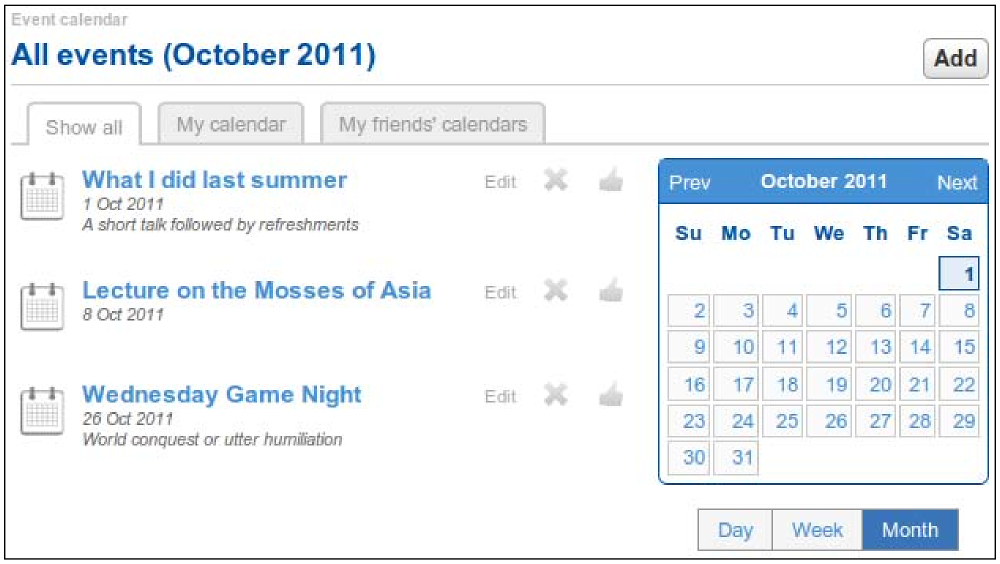
- Site-wide and group calendars
- Track who is interested in an event
- Profile widget for upcoming events
- Extensive configuration options

Administration

The event calendar provides an impressive number of administrative options for tailoring it to a site. The administrator controls who can create events, whether groups get their own calendars, what fields are required, and how events should be displayed. There are over 30 different configuration parameters.

Site calendar

When creating an event, a user can set the title, description, location, date, time, cost, and several other fields. Every event is added to a site-wide calendar. This calendar can be viewed by month, week, or day.



What I did last summer
1 Oct 2011
A short talk followed by refreshments

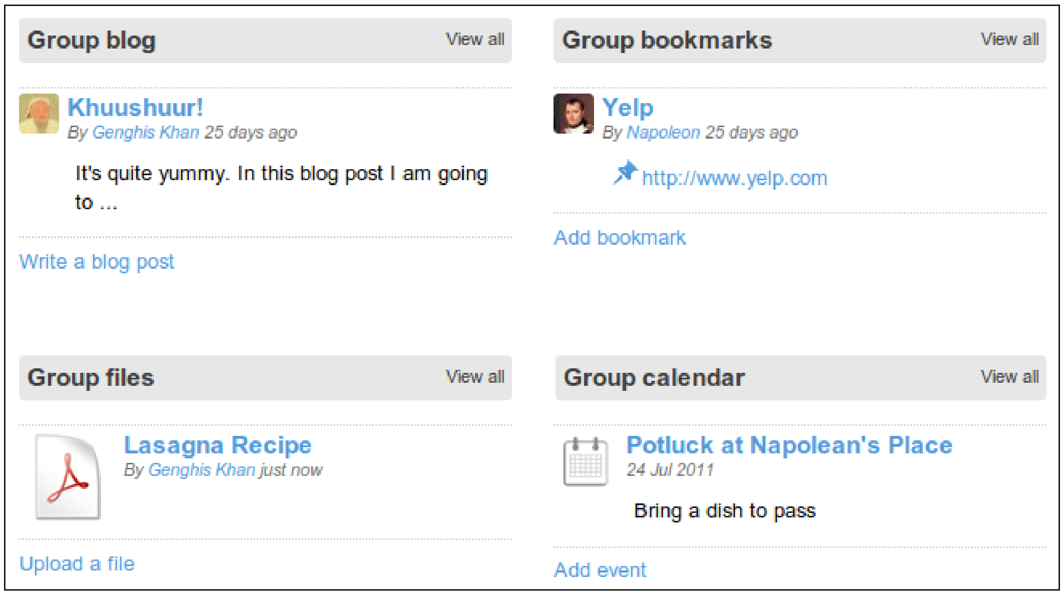
Lecture on the Mosses of Asia
8 Oct 2011

Wednesday Game Night
26 Oct 2011
World conquest or utter humiliation

Users can add events to their personal calendar. They have a web page for tracking what events are on their calendar, or they can add a personal calendar widget to their profile pages. There is also a web page available to everyone to see who has added an event to a personal calendar.

Group calendar

If this feature is enabled, then each group has its own calendar. It works exactly like the site calendar and also includes an upcoming events widget for the group profile page as shown in the following screenshot. Group events can be automatically added to group members' personal calendars if the site administrator enables that option.



Profile Manager

The Profile Manager plugin provides an impressive amount of control over a user's profile while also integrating with user registration.

Plugin profile

Author: Jeroen Dalsem of ColdTrick IT Solutions

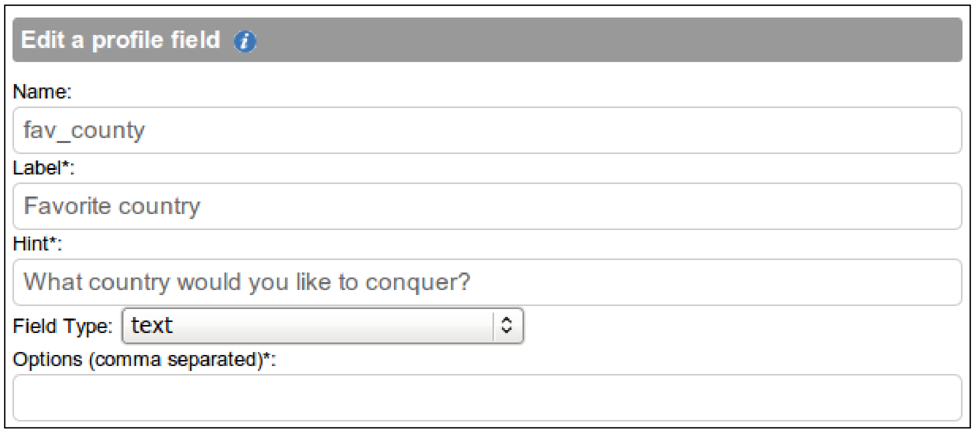
Link: `http://community.elgg.org/pg/plugins/project/385114`

Features include:

- Additional profile field type (dates, files, and more)
- Profile field categories
- Mandatory fields
- Read-only fields
- Collect profile information during registration
- Custom search based on profile information
- Ajax administrative interface

Adding profile fields

The Profile Manager has a simple interface for adding new profile fields. The administrator enters the label and selects one of 11 different profile field types. If the field has a limited set of values, then they are entered as a comma-separated list.



Edit a profile field

Name: fav_county

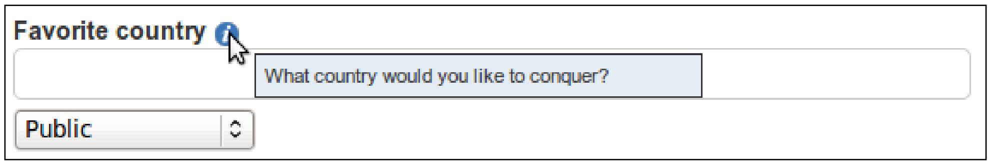
Label*: Favorite country

Hint*: What country would you like to conquer?

Field Type: text

Options (comma separated)*:

Another feature of the Profile Manager is the ability to add a hint for each profile field. It shows up as a tooltip when a user is filling in the profile fields, as shown in the following screenshot:



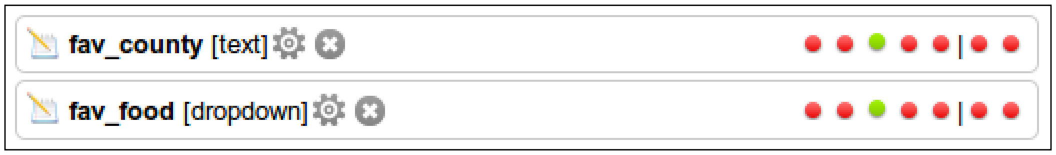
Favorite country

What country would you like to conquer?

Public

Configuring the profile fields

Beyond creating new fields, this plugin also offers several configuration options per profile field. The red and green lights for each profile field indicate whether or not the options are enabled. The options include whether the field is shown on the registration field, whether it is mandatory, and whether the profile field is editable by the user. This last option is very useful if you are automatically populating some of the fields from a database.

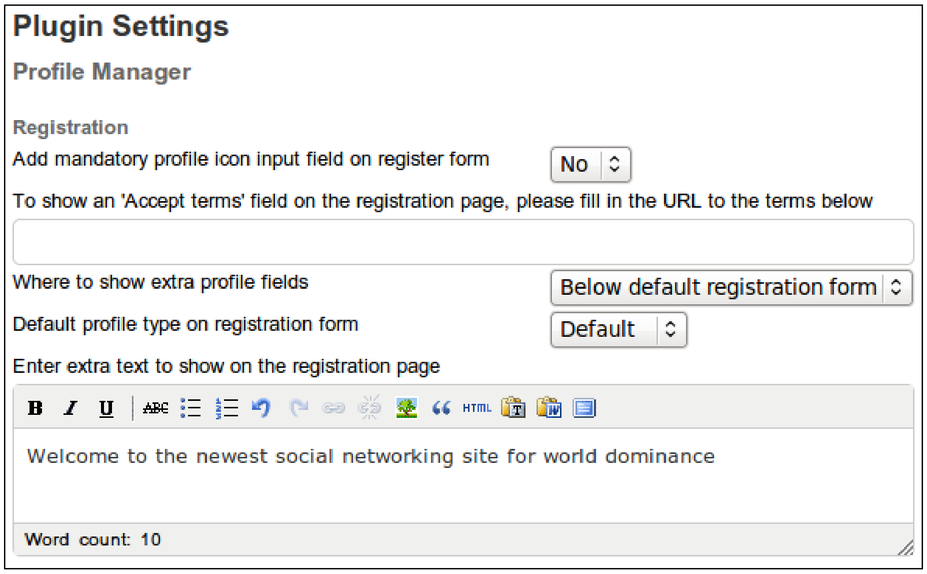


fav_county [text] ● ● ● ● ● | ● ●

fav_food [dropdown] ● ● ● ● ● | ● ●

Plugin settings

As with the other plugins described in this chapter, the Profile Manager supports customization through plugin settings. The Administrator can require that an avatar be uploaded during registration, add a special member's search page, or include a link to the site's terms on the registration page.



Plugin Settings

Profile Manager

Registration

Add mandatory profile icon input field on register form: No

To show an 'Accept terms' field on the registration page, please fill in the URL to the terms below

Where to show extra profile fields: Below default registration form

Default profile type on registration form: Default

Enter extra text to show on the registration page

Welcome to the newest social networking site for world dominance

Word count: 10

Summary

You should now feel comfortable installing and configuring plugins. We covered how to find, evaluate, and install plugins. We also discussed how themes are plugins and where to find them. The plugins reviewed in this chapter are three of the top plugins available in the community repository. Not only are they popular, but they offer more capabilities and more configuration options than almost all of the other available plugins. Now is a great time to try out some more plugins before you begin writing your own plugins in the next chapter.

7
Creating Your First Plugin

The next three chapters are a beginner's guide to customizing Elgg. This chapter leads you through the process of creating your first plugin. It introduces you to several components of Elgg's plugin API and provides advice on debugging plugins. *Chapter 8, Customization through Plugins*, completes the introduction of plugin development through a series of examples that teach different facets of Elgg development. An overview of theming is presented in *Chapter 9, Theming Elgg*, to wrap up this section of the book.

These chapters are tutorial in nature and slowly build up your knowledge of plugin development. They do not cover everything there is to know about Elgg. *Appendix A, Developer's Quick Start Guide*, provides a systematic look at the Elgg framework. It complements the learning-by-example approach of these chapters. Your starting point depends on your skill level and your preference for tutorials versus high-level overviews.

The plugin code is available for download at `http://www.packtpub.com/elgg-18-social-networking/book`. The combination of the explanations in this chapter and the comments in the code will provide you with the information needed to modify the plugins for your own use. The plugins are examples of common customizations.

The topics covered in this chapter are as follows:

- Setting up your development environment
- Creating a plugin skeleton
- Using the Elgg event system
- Adding new web pages
- Creating views to render content into HTML
- Using the menu system
- Supporting multiple languages
- Debugging a plugin

What you need to know

This chapter assumes that you have at least basic software development skills and experience with PHP and HTML. You do not need to be an expert in those languages. You should be able to read and understand their basic syntax and feel comfortable looking up PHP functions and HTML markup that are unfamiliar to you. To be an expert in creating Elgg plugins, a developer needs to know PHP, HTML, CSS, and JavaScript (especially the jQuery library). Most web developers are good in two or maybe three of those languages and know enough in the others to get by.

If you have never created a web page before or do not know what a `for` loop is, then this chapter will be be difficult for you. You will need to learn the basics first. A good starting place is introductory books on HTML and PHP or working through tutorials on the Web.

The last thing that you need to know is how PHP works with the web server to create web pages. This is a common point of confusion with developers who are new to web development. Let's walk through the process of creating a web page in PHP: A web browser requests a page from the web server.

1. The PHP code is loaded and compiled.
2. The PHP interpreter runs the compiled code.
3. The code produces a web page as a long string. This string is sent by the web server back to the web browser.

The web browser renders that string into a web page, including requesting resources such as images and style sheets. When another request is received, the same process happens again. This is very different from a desktop application like a word processor that continues to run as a user interacts with it. The PHP application shuts down between requests from a user. To maintain state, the application uses session cookies or stores information in a database. This has implications for building and debugging Elgg plugins (for example, you cannot set a variable when creating one web page and expect to access it on another web page without saving it to the database).

Elgg developer resources

Several resources for developers were briefly mentioned in the first chapter. Here they are with more detail:

- Wiki documentation (`http://docs.elgg.org/`). The Elgg wiki has an overview of the framework, tutorials, frequently asked questions, and information on contributing to the Elgg project. It also has a list of commonly used functions in plugin development. This list is also available as a PDF from the Keetup development team at `http://www.keetup.com/download/elgg-cheatsheet.pdf`.
- API Reference (`http://reference.elgg.org`). All the documentation in the code is extracted and made available in this web resource. Its best use is searching for functions.
- Coding standards and best practices. Elgg has its own coding standard and a tool for validating code against that standard (`https://github.com/cash/elgg-coding-standards`). The coding standards and best practices are included with the source code in the documentation directory.
- Google Group for developers (`http://groups.google.com/group/elgg-development`). There are between 50-100 messages a month in this group. Topics range from introductory (*how do I...*) to involved discussions about the future features of the Elgg engine.
- Community forums (`http://community.elgg.org`). The forums have a mix of experience and skill levels. If you are new to PHP and software development, then this is probably a better place to ask beginner questions. There is also significantly more activity in the forums than the developer group.
- IRC channel (#elgg on Freenode). The core developers hold status meetings in the #elgg channel. Activity varies greatly.
- Elgg blog (`http://blog.elgg.org`). A great resource for keeping up with new releases and development work on the Elgg engine.
- Elgg Github account (`https://github.com/Elgg`). Elgg core development team uses Github as a repository for the Elgg engine and many additional plugins. The easiest way to contribute a bugfix or patch is through Github's pull request feature.
- Bug tracker (`http://trac.elgg.org`). If you think you found a bug or have an idea to improve Elgg, then this is the place to report it.

Setting up your development environment

To write plugins, you need an editor to work with the code, and you need to configure your site to support plugin development.

Editing code

At a minimum, you will want an editor that is built for working with the programming languages that Elgg uses. This could be a text editor such as Textpad++ on Windows or TextWrangler on Macs. (No, Microsoft Word and Notepad are not options.) The text editor should have features such as syntax highlighting and searching through files for a particular pattern.

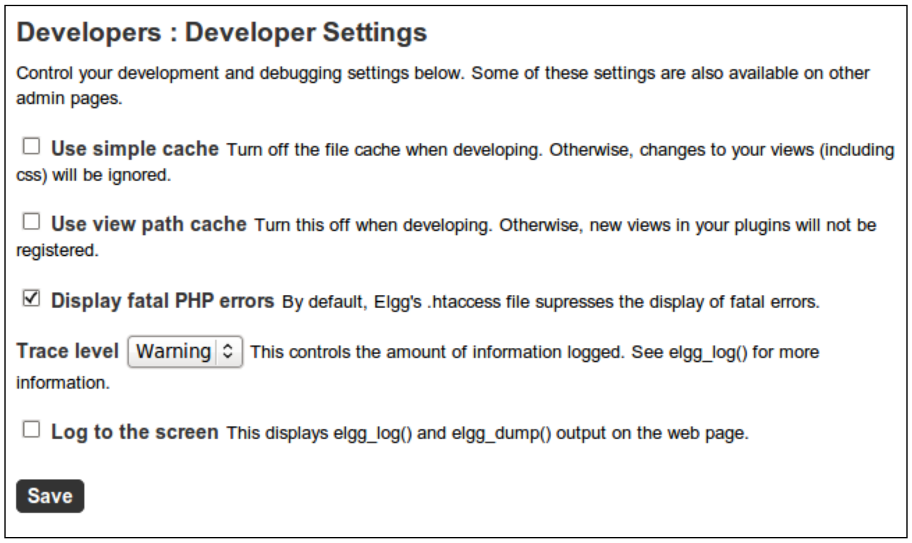
A better option, especially for those new to PHP or Elgg, is an **Integrated Development Environment** (**IDE**). An IDE is a collection of tools for working with code. With an IDE, you can edit, run, debug, and test code. Important features of most IDEs are as follows:

- **Syntax checking**: Why wait to test your code on the server to find out that you forgot to add a semi-colon to a line? IDEs can check your code as you type it and tell you when it is invalid.
- **Code navigation**: IDEs enable you to jump from where a function is called to where it is defined and vice-versa.
- **Code completion**: Start typing a function or variable name and the IDE will provide the possible matches. It saves time and helps those new to a framework to learn its functions and methods.
- **Documentation hints**: An IDE can show you the documentation for a function or class anywhere it is used.
- **Auto formatting**: Elgg has a coding standard to help keep code clean and readable. You can configure an IDE to automatically format code according to the standard.
- **Debugging**: IDEs have a debugger for interactively hunting for bugs.

Two popular open source IDEs are NetBeans (`http://netbeans.org/features/php/`) and Eclipse (`http://www.eclipse.org/pdt/`). All the IDE screenshots in this book are of NetBeans with the PHP extension.

Configuring your site

There are a few modifications that we need to make to your test site before we start working with code. First, activate the **Developers** plugin if you have not already. It adds a **Develop** section to the administration sidebar menu with some very useful tools. Select the **Developer Settings** page from the menu.



Developers : Developer Settings

Control your development and debugging settings below. Some of these settings are also available on other admin pages.

☐ **Use simple cache** Turn off the file cache when developing. Otherwise, changes to your views (including css) will be ignored.

☐ **Use view path cache** Turn this off when developing. Otherwise, new views in your plugins will not be registered.

☑ **Display fatal PHP errors** By default, Elgg's .htaccess file supresses the display of fatal errors.

Trace level Warning This controls the amount of information logged. See elgg_log() for more information.

☐ **Log to the screen** This displays elgg_log() and elgg_dump() output on the web page.

We need to make three changes:

1. **Turn caching off**. To speed up the creation of your web pages, Elgg stores generated content rather than recreating it each time. If caching is turned on, then Elgg may ignore changes to a plugin. Turn off both simple cache and the view path cache.
2. **Turn display of fatal errors on**. This may be stating the obvious, but PHP dies when it experiences a fatal error. In Elgg's `.htaccess` file, the display of these error messages is suppressed. This prevents users from seeing overly technical error messages if something goes wrong. We want to see those errors in our browser while we write and test code, so check this option.
3. **Set the trace level to Warning**. We want Elgg to write any error or warning to the server's log while we are working with code. We will discuss this logging in the section on debugging. Furthermore, we do not set it to Notice as that writes all the database queries to the log, making it difficult to find information.

When you are done setting those options, save the settings. Now we are ready to start coding.

Hello, World!

A `Hello, World` program is the classic beginner's program in any language. In it, a developer writes code to print out the words *Hello, World*. It is often the simplest program that can be written in a language and provides a sanity check by proving that a program can be compiled and executed. We are borrowing this concept to create a `Hello, World` plugin for Elgg. The plugin creates a web page that displays a greeting. We then add more functionality to it as we explore different parts of Elgg's plugin API. This tutorial introduces you to the following:

- Setting up a basic plugin skeleton
- Using Elgg's event system
- Routing requests for a web page
- Creating a new web page
- Supporting multiple languages (in developer's slang, I18n or internationalization)
- Registering menu items
- Creating views that render data into HTML

All the tutorials in this book are written for Elgg 1.8. They will not work on earlier versions of Elgg because they use functions introduced in Elgg 1.8. If any of the plugins use a function that is deprecated in a future release of Elgg, then they will work for two subsequent major releases. Using a deprecated function produces a warning message that tells the developer how to update the code. The deprecation process is documented in `/documentation/coding_standards/deprecation.txt`.

Plugin skeleton

All the most basic plugin requires is a directory and two files. Let's get started.

Create your plugin directory

Plugins live in the `/mod` directory within Elgg. Each plugin has its own directory. The directory name should describe the plugin's function. For our plugin, we will create a directory called `hello_world`.

Plugin manifest

The `manifest.xml` file gives Elgg details about the plugin to display on the plugin page in the administration area. The required fields are plugin name, author name, plugin version, description, and the Elgg release that the plugin is created for. For more information on plugin manifests, read `http://docs.elgg.org/Manifests`.

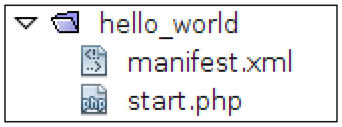
Copy the manifest file from `/documentation/plugins/skeleton` into the `hello_world` directory and then edit it so that it describes this plugin. It should look similar to the following code:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
    <name>Hello, World</name>
    <author>Cash Costello</author>
    <version>1.0</version>
    <description>My Hello, World! Plugin</description>
    <website>http://www.elgg.org/</website>
    <copyright>Cash Costello 2011</copyright>
    <license>GNU General Public License version 2</license>
    <requires>
        <type>elgg_release</type>
        <version>1.8</version>
    </requires>
</plugin_manifest>
```

Start script

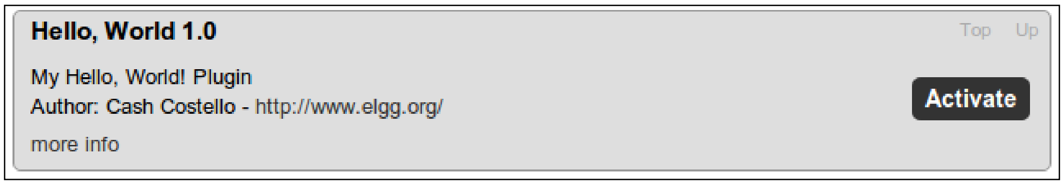
When you press the power button on your computer, it boots up. This boot process gets the computer ready for you to write e-mails or use the web. Elgg also has a boot sequence that loads each activated plugin's start script.

Create an empty file named `start.php` in `hello_world`. The plugin skeleton looks like the following:



Activating the hello world plugin

Go to the **Plugin** page in the administration area of your test site. The `Hello, World` plugin should be listed last on this page as it is the newest plugin.



Hello, World 1.0

My Hello, World! Plugin
Author: Cash Costello - http://www.elgg.org/
more info

Activate it. This is your first plugin (even though it does not change the site yet).

Register for the init, system event

Every time a web page is requested from an Elgg site, the Elgg start script runs (located at `engine/start.php`). The overall sequence of the script is as follows:

1. Load Elgg's libraries and configuration settings.
2. Load the plugin's start scripts.
3. Trigger the `init`, `system` event.

The first two stages load code, connect to the database, and load configuration information. Then, the `init`, `system` event indicates that the loading has finished and it is time to run initialization functions. During initialization, a plugin tells the engine what capabilities it has and when its functions should be called. For example, a plugin can ask to be called whenever a comment is posted on the site or the plugin can add an item to the site menu. Once initialization finishes, Elgg begins to handle the request for the web page.

In order for the `Hello, World` plugin to run during initialization, we need to register for the `init, system` event. This is done through the `elgg_register_event_handler()` function. Add this code into the plugin's empty `start.php` file:

```
<?php
/**
 * Hello world plugin
 */

  elgg_register_event_handler('init', 'system',
    'hello_world_init');

  function hello_world_init() {
  // do nothing right now
}
```

When Elgg loads the plugin's `start.php` file, `elgg_register_event_handler()` is executed, which registers the function `hello_world_init()` for the `init, system` event. `hello_world_init()` does not do anything yet, but we will be adding to it throughout this tutorial.

The `init, system` event is one of many available in Elgg's event system. Plugins use the event to register callback functions that run whenever certain things occur. The events could be a user logging in, a blog post being published, or a user updating her profile. For more information on events, read `http://docs.elgg.org/Events_Overview`. A list of events is available at `http://docs.elgg.org/List_of_Events`.

Adding a new route

Now that we have an initialization function in our plugin, we want to start receiving requests for web pages. This is done by registering with Elgg's routing system. The routing system works by checking the first segment of a URL against a list of registered page handlers. A **page handler** is a function that responds to a request from a web browser (in MVC terminology, it is a controller). As an example, if Elgg receives a request from a web browser for Brett's latest blog posts:

`http://example.org/blog/owner/brett`

it calls the page handler registered for the `blog` identifier. The other segments of the URL are passed to the page handler in an array. The blog page handler in this example receives the following array:

```
$page = array('owner', 'brett');
```

Based on these URL segments, the page handler gathers the appropriate data and gives it to the view system to render into a web page.

Register for the identifier `hello` with our `Hello, World` plugin. The `start.php` file looks like the following code snippet:

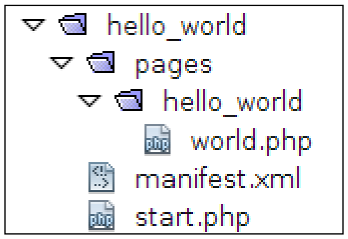
```
<?php
/**
 * Hello world plugin
 */

elgg_register_event_handler('init', 'system', 'hello_world_init');

function hello_world_init() {
  // register to receive requests that start with 'hello'
  elgg_register_page_handler('hello', 'hello_page_handler');
}

function hello_page_handler($page, $identifier) {

  echo "request for $identifier $page[0]";

  // return true to let Elgg know that a page was sent to browser
  return true;
}
```

In `hello_world_init()`, we added a call to `elgg_register_page_handler()`, which tells Elgg to call the function `hello_page_handler()` when a request starts with `hello`. If a request is received for `http://example.org/hello/world`, then the `echo` statement outputs the string *request for hello world* back to the browser. Try replacing `world` with other words or add another URL segment and display that one, too.

Creating a web page

People will think our website is broken if our plugin outputs plain text back to the web browser. In this step, we create a script to handle a request and produce HTML. It is also our first introduction to Elgg's view system, which is responsible for rendering data into HTML.

In your plugin's directory, create a sub-directory named `pages`. In the `pages` directory, create the sub-directory `hello_world`. Finally, create a file in this directory named `world.php`. The plugin structure now looks like the following:



Neither the `pages` nor the `hello_world` directories are required, but they are recommended by the Elgg developers as providing good organization and support for future features.

The `world.php` file has the code to create a web page, as follows:

```
<?php
/**
 * hello, world page
 */
```

```
$title = "My first page";
$content = "Hello, World!";

$vars = array(
  'content' => $content,
);
$body = elgg_view_layout('one_sidebar', $vars);

echo elgg_view_page($title, $body);
```

First, we set the title and the content of the page. Next, we pass the content into `elgg_view_layout()` through the associative array `$vars`. (Passing variables this way is standard with the views system. We will be discussing this later in the chapter.) The layout function takes our content and positions it within a layout that has a main content area and a single sidebar. Finally, `elgg_view_page()` assembles the entire web page and the echo statement sends it to the browser.

Update routing

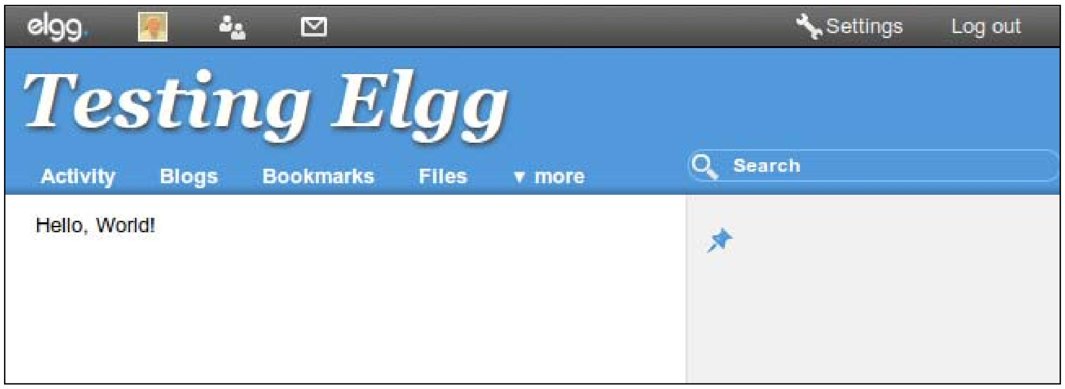
With this script we can create a web page, but we still need to connect the script to our page handler. This is done in the page handler by including the script, as follows:

```
function hello_page_handler($page, $identifier) {

  $plugin_path = elgg_get_plugins_path();
  $base_path = $plugin_path . 'hello_world/pages/hello_world';

  // select page based on first URL segment after /hello/
  switch ($page[0]) {
    case 'world':
      require "$base_path/world.php";
      break;
    default:
      echo "request for $identifier $page[0]";
      break;
  }

  // return true to let Elgg know that a page was sent to browser
  return true;
}
```

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.PacktPub.com`. If you purchased this book elsewhere, you can visit `http://www.PacktPub.com/support` and register to have the files e-mailed directly to you.

If a request is received for `http://example.org/hello/world`, then the page handler includes the `world.php` script that we just created and executes the code. This sends our new web page to the requester:



Hello, World!

Besides sending an actual web page to look at, we also set the title of the page in `elgg_view_page()`. When you view the page, the title appears in the browser title bar or tab.

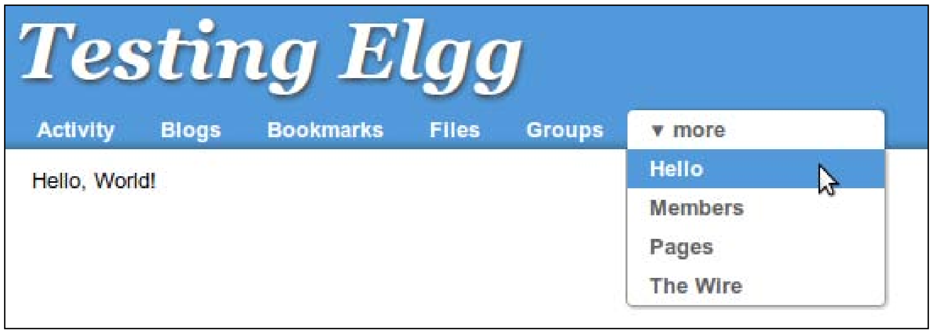
To experiment, try changing the layout. Besides `one_sidebar`, Elgg also has `one_column` and `two_sidebar` layouts. The layout view files are located with Elgg's directory structure at `/views/default/page/layouts/`. The top of each file lists what parameters it accepts. For example, to add text to the sidebar use a key of `sidebar` in the `$vars` array.

Add to the site menu

We have one more step before visitors to the site can find our new web page: adding it to the site menu. In the default theme, the site menu is the set of tabs above the content area. We register an item for the menu in our initialization function, as follows:

```
  function hello_world_init() {

   elgg_register_page_handler('hello', 'hello_page_handler');

   // add a menu item to primary site navigation
   $item = new ElggMenuItem('hello', 'Hello', 'hello/world');
   elgg_register_menu_item('site', $item);
 }
```

When we create the `ElggMenuItem` object, we pass the name, display text, and the URL we want the menu item to link to. The name must be a unique identifier for that menu. The URL can be partial as in our example. Elgg automatically appends the site's canonical URL to addresses that it detects as partial. The last line of code adds the menu item to the site menu so that it looks like the following screenshot:



Extending the page handler

Our last modification to the page handler is adding another web page so that we have two pages:

- `http://example.org/hello/world/`
- `http://example.org/hello/dolly/`

The second page will display *Hello, Dolly!* You may want to stop reading now and try adding this page without the instructions as a test of your new knowledge. If not, then the updated page handler looks like the following code snippet:

```
function hello_page_handler($page, $identifier) {

  // file path to the page scripts
  $plugin_path = elgg_get_plugins_path();
  $base_path = $plugin_path . 'hello_world/pages/hello_world';

  // select page based on first URL segment after /hello/
  switch ($page[0]) {
    case 'world':
      require("$base_path/world.php");
      break;
    case 'dolly':
      require("$base_path/dolly.php");
      break;
    default:
            echo "request for $identifier $page[0]";
            break;
    }

    // return true to let Elgg know that a page was sent to browser
    return true;
}
```

Add a second file to the `pages/hello_world` directory named `dolly.php` and put this code in it:

```
<?php
/**
 * hello, world page
 */

$title = "My second page";
$content = "Hello, Dolly!";

$vars = array(
  'content' => $content,
);
$body = elgg_view_layout('one_sidebar', $vars);

echo elgg_view_page($title, $body);
```

When you point your web browser to the new URL, you should see the welcome for Dolly.

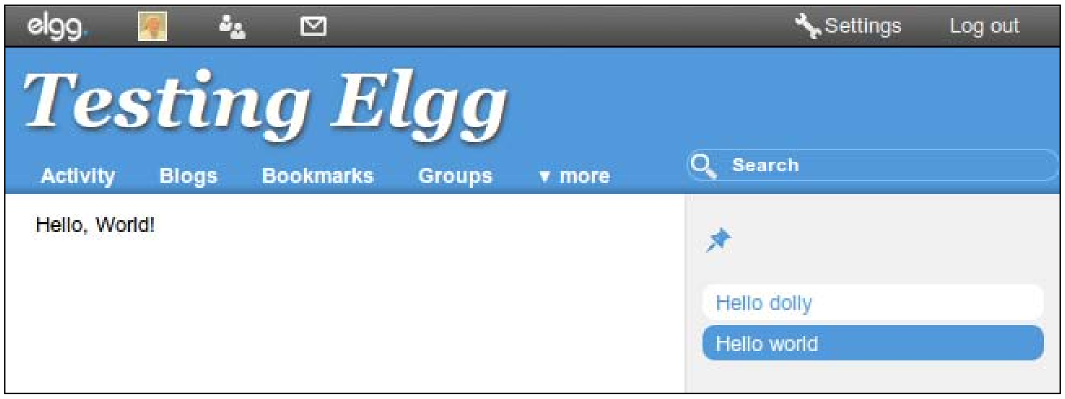
Add a sidebar menu

Users can navigate to one of the pages, but how do they find the other one? In this section, we add a sidebar menu so that users can choose between the two different hello pages. To create the sidebar menu, we register our menu items in the plugin's initialization function.

```
function hello_world_init() {

  elgg_register_page_handler('hello', 'hello_page_handler');

  // add a menu item to primary site navigation
  $item = new ElggMenuItem('hello', 'Hello', 'hello/world');
  elgg_register_menu_item('site', $item);

  // add sidebar menu items that only show up on 'hello' pages
```

```
        elgg_register_menu_item('page', array(
          'name' => 'world',
          'text' => 'Hello world',
          'href' => 'hello/world',
          'contexts' => array('hello'),
        ));
        elgg_register_menu_item('page', array(
          'name' => 'dolly',
          'text' => 'Hello dolly',
          'href' => 'hello/dolly',
          'contexts' => array('hello'),
        ));
      }
```

This time we use an alternate method of calling `elgg_register_menu_item()` by passing an associative array rather than an `ElggMenuItem` object. The end result is the same, but it can be easier to set several parameters with the array approach. In this case, we are not only setting the name, text and URL, but also configuring the context for the menu items. Setting the context to `'hello'` results in these menu items only displaying when we are serving requests that start with `'hello'`. If we changed the contexts value to `array('hello', 'blog')`, the menu items would display on hello and blog pages. To read more on context, visit `http://docs.elgg.org/Context`. For more information on the menu system, visit the Elgg blog (`http://blog.elgg.org`) and read the series of articles on using the menu system.



If you view one of the hello pages, you should see the new menu items in the sidebar as shown in the preceding screenshot.

You need a sidebar to have a sidebar menu
If you modified the layout of the pages earlier, then return them to using the `'one_sidebar'` layout. With the `'one_column'` layout, there is nowhere for the sidebar menu to be displayed.

Language support

Our plugin only supports the English language. Adding support for other languages requires two modifications:

1. Use `elgg_echo()` instead of hard-coded strings.
2. Add a language file.

Anywhere we used a string that is displayed, we are replacing it with a call to the `elgg_echo()` function. For example, in `world.php`:

```
$content = "Hello, World!";
```

This should be changed to the following:

```
$content = elgg_echo('hello:world');
```

The function `elgg_echo()` takes in a descriptor string and outputs the appropriate string in the user's chosen language. The descriptor strings should describe the text and its purpose. For example, if you had a long welcome message on your page, the descriptor string could be `'hello:welcome:msg'`. You can use underscores, hyphens, or colons to separate words in the descriptor strings. For readability, avoid stringing together words such as this: `'thisismydescriptorstring'`. Another best practice is to namespace the descriptor strings based on the plugin. This means you should append an identifier to all the descriptors as in the following language file. Doing this prevents two plugins from using the same descriptor with different definitions.

```
<?php
/**
 * English language file
 */

$mapping = array(
   'hello:world'      => "Hello, World!",
   'hello:dolly'      => "Hello, Dolly!",
  'hello:menu'      => "Hello",
  'hello:sidebar:world'  => "Hello world",
  'hello:sidebar:dolly'  => "Hello dolly",
);

add_translation('en', $mapping);
```

As you gain more experience and look at other plugins, it will become easier to select the descriptor strings. Continue the replacement for any string that appears on the plugin's web pages using the preceding code for the descriptor strings. After the replacement is finished, create the directory `languages` in your plugin's main directory. In the languages directory create a file named `en.php` and copy the preceding code into it.

This file creates an array that maps from the descriptor string to the display string. When `elgg_echo('hello:world')` is called, it outputs *Hello, World!*. If we wanted to include French on the site, then we would create a file called `fr.php`, create the mapping for that language, and then call `add_translation()`, specifying the language code as `'fr'`. The two letter codes come from the ISO 639-1 short code definition. Elgg's primary language file (`/languages/en.php`) contains a long list of these codes at the bottom of the file.

Personalizing the content

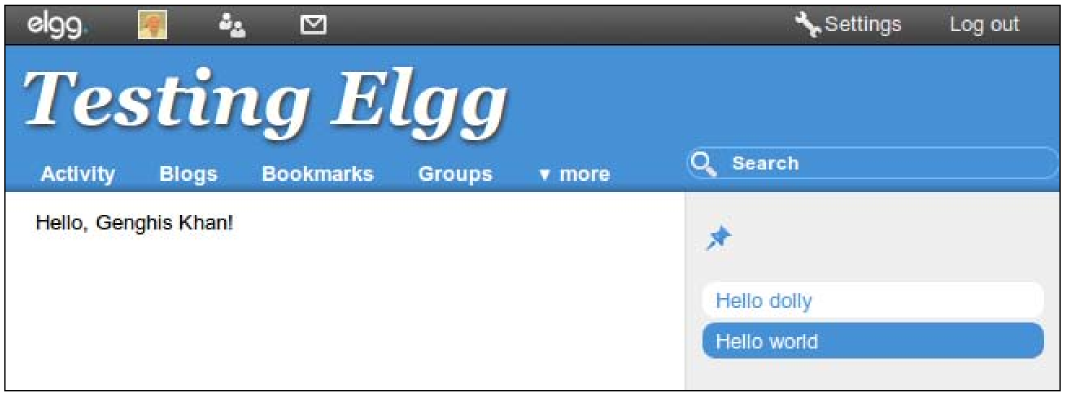
A generic "*Hello, World*" greeting is not exactly a warm welcome to the site. Let's make the greeting more personalized by using the user's name. Modify `world.php` to look like the following code snippet:

```
<?php
/**
 * hello, world page
 */

$user = elgg_get_logged_in_user_entity();

$title = "My first page";

$content = elgg_echo('hello:user', array($user->name));

$vars = array(
  'content' => $content,
);
$body = elgg_view_layout('one_sidebar', $vars);

echo elgg_view_page($title, $body);
```

The first line of code gets the logged in user. That gives us access to all sorts of information about the user such as: name, e-mail address, last login date, join date, profile icon, and more. We are using the user's display name in the greeting message. The `elgg_echo()` function works like `sprintf()` and can accept parameters that are inserted into the outputted string. Update the `'hello:user'` definition in `en.php` to be as follows:

```
'hello:user' => "Hello, %s!",
```

so that `elgg_echo()` replaces `%s` with the user's name. The output of the page looks like the following screenshot:



Hello, Genghis Khan!

To experiment more with Elgg's API, replace the welcome text with the user's icon using `elgg_view_entity_icon()`. When `$user` is passed to this function, it returns the HTML for the icon.

Organizing your content into views

Think about what the code for `world.php` would look like if we added a few images, a list of important information, a site logo, and contact information for the site administrator. The code would become very complicated and probably very ugly.

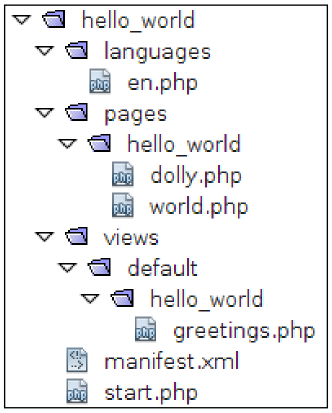
We may also want to have the same content appear on multiple web pages. Maybe the site logo and contact information belongs on every page. To do this, we could copy that portion of the code into all the pages. When we want to make one change to that content, we would have to change every file. This is beginning to look painful.

There is a solution. Elgg has a views system that enables us to break our content into sections and then include it in our pages with a single function call. Each view is a chunk of HTML code. This is how Elgg creates the topbar, header, sidebar, and footer on each page. There is a view for the topbar, a view for the header, and so on.

A greeting view

Both of our pages contain a greeting. In `world.php`, it is currently a personalized greeting. In `dolly.php`, it is a hard-coded greeting for Dolly. Let's create a single view that can handle both of these uses.

HTML views live in the `views/default` directory of a plugin. Create the `views` directory and then create the `default` directory underneath `views`. It is a good idea to store your views in a directory named after your plugin to prevent conflicts with other plugins. Create a directory called `hello_world` in the `default` directory. Create the file `greetings.php` in `hello_world` so that the directory structure looks like the following image:



```
hello_world
  languages
    en.php
  pages
    hello_world
      dolly.php
      world.php
  views
    default
      hello_world
        greetings.php
  manifest.xml
  start.php
```

Open `greetings.php` and add this to it:

```
<?php
/**
 * Content area greeting
 *
 * @uses $vars['name'] The name of a user
 */

echo elgg_echo('hello:user', array($vars['name']));
```

Then, update `world.php` to look like the following code snippet:

```
<?php
/**
 * hello, world page
 */
```

```
$user = elgg_get_logged_in_user_entity();

$title = "My first page";

$params = array('name' => $user->name);
$content = elgg_view('hello_world/greetings', $params);

$vars = array(
  'content' => $content,
);
$body = elgg_view_layout('one_sidebar', $vars);

echo elgg_view_page($title, $body);
```

There are a few things to note about these changes.

First, we are using an associative array to pass the name of the user to the `'hello_world/greetings'` view. View files always access the variables passed to them through the `$vars` array. We document the variable that this view accepts at the top of the file with the `@uses` tag. Most of Elgg's views have similar documentation.

Second, the view was located in our plugin at `views/default/hello_world/greetings.php` and is named `'hello_world/greetings'`. The name of the view always starts after the default directory.

Third, the call to `elgg_view()` returns the output of the view as a string. All of the functions that begin with `elgg_view` do this.

What is "default"?

There are different types of views in Elgg. The `default` views produce HTML. That is why you created a directory under views called default. There are other views that produce RSS feeds or JSON data. Elgg knows what views to use based on the request. This enables plugins to use the same handler logic for different types of outputs. For further reading on Elgg's view system and view types in particular, visit `http://docs.elgg.org/Engine/Views`.

To update `dolly.php` to use the view, change the following:

```
$content = "Hello, Dolly!";
```

To the following:

```
$vars = array('name' => 'Dolly');
$content = elgg_view('hello_world/greetings', $vars);
```

The output of the pages has not changed, but if we wanted to change the text on both pages to be a heading, we would only have to change a single view. Change the greeting view (`hello_world/views/default/hello_world/greeting.php`) to the following code snippet:

```
<?php
/**
 * Content area title
 *
 * @uses $vars['name'] The name of a user
 */

$greeting = elgg_echo('hello:user', array($vars['name']));
echo elgg_view_title($greeting);
```

The greetings on both pages are much larger now.

A stats view

There is a lot of blank space on those pages so we are adding another view that tells the user how many blog posts he has written. In `world.php`, we calculate the number of posts and pass it to a new `'hello_world/blog_stats'` view. The calculation code uses `elgg_get_entities` like the following code snippet:

```
// count number of blogs by user
$options = array(
  'type' => 'object',
  'subtype' => 'blog',
  'owner_guid' => $user->guid,
  'count' => true,
);
$num_blogs = elgg_get_entities($options);

// add the stats view to the content for page
$params = array('num_blogs' => $num_blogs);
$content .= elgg_view('hello_world/blog_stats', $params);
```

Elgg provides a set of functions for getting and displaying information about content from the database and `elgg_get_entities()` is one of them. There will be more information about these functions in the next chapter.

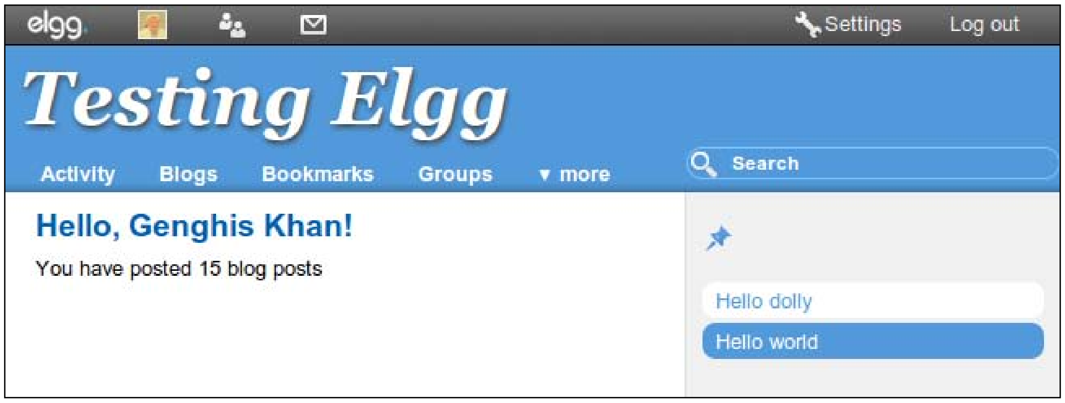
What is a GUID?

GUID stands for **Globally Unique IDentifier**. It is a numerical value assigned to every user, group, blog post, or any other kind of content. No two entities in Elgg's database share the same GUID. In the stats view, we passed the GUID of the user to get a count of blog posts.

Add a `blog_stats.php` file to the `views/default/hello_world` directory and add the following code:

```
<?php
/**
 * Example view that displays some blog stats
 *
 * @uses $vars['num_blogs'] Number of blogs a user has published
 */
?>
<p>
<?php echo elgg_echo('hello:blog:stats', array($vars['num_blogs']));
?>
</p>
```

After adding the `'hello:blog:stats'` string to the language file, the end product will look like the following screenshot:



Hello, Genghis Khan!

You have posted 15 blog posts

Now that you have this basic plugin skeleton, feel free to experiment by creating additional views or accessing different data about the user. For a list of variables that are available for a user, see `http://docs.elgg.org/ElggUser`.

Review

We covered a lot of content in this example plugin. We introduced four key components of Elgg, as follows:

1. **Events**: A plugin can register a function to be called when something happens, like when a user posts a comment. We used the event system for plugin initialization.
2. **Page handling**: Elgg handles routing requests to registered page handler functions and provides the segments of the requested URL. A plugin's page handler function loads the code that creates the web page.
3. **Views**: Views are chunks of HTML that can be reused. They provide a means of organizing your presentation code.
4. **Multi-lingual support**: By defining a language mapping and using `elgg_echo()` for all displayed text, plugins can support multiple languages.

More information about views and events are included in the next chapter. We will also learn about Elgg's action system, which is the primary method for plugins to add new content to the database.

Before we work through more examples in the next chapter, we will cover information about debugging plugins. How quickly you can find and fix a plugin's bugs may be the biggest factor on how long it takes to write a plugin.

Debugging

Programmers develop good coding habits to reduce the number of bugs in their code. They never eliminate them. Good debugging skills are a result of experience, a solid understanding of how the software works, logical thinking, and dogged persistence. If you have limited experience debugging code, then the process is a lot like how auto mechanics work:

1. Collect symptoms of the problem.
2. Match symptoms to possible causes.
3. Begin eliminating potential causes.
4. Repair most probable cause.
5. Test the repair to confirm that the problem is fixed.

The more you work through this process, the more it will become second nature to you.

This section introduces the tools that are available for debugging Elgg plugins. Debugging plugins can be challenging because there can be problems on the server or the client and the bug can be in PHP, HTML, CSS, or JavaScript code. Because of this, a good web developer is comfortable with a set of tools that span these languages: debuggers, web browser developer tools, and various logging techniques.

Debugging to the log

The web server on your server probably writes information to two log files on a regular basis: an access log that tracks the requests made for web pages and an error log that records when something goes wrong. By default, PHP sends its logging information to the web server's error log file. Locate the error log on your server and glance through it. For help with finding the error log, see `http://docs.elgg.org/Server_error_log`. If there are PHP messages in the log, then they will include the level of the message (error, warning, notice), the text of the message, and the line number and name of the file where the problem occurred.

```
[Fri Jul 01 23:40:02 2011] [error] [client 192.168.1.101] PHP
  WARNING: 2011-07-01 23:40:02 (EDT): "Invalid argument supplied for
  foreach()" in file
  /var/www/mod/developers/views/default/developers/log.php (line 10)
```

To write something to the log, use the PHP function `error_log()`. Test this out by adding the following statement to the `world.php` file in the `Hello, World` plugin:

```
error_log('Creating world page');
```

View the page with your browser and then check the error log again. There should be a new entry in the log that contains the text *Creating world page*. Putting in a debugging statement like this is a sanity check. You are verifying that the page or function with the logging call is actually being called.

Anything that can be turned into a string can be passed to `error_log()`. In the same file, try the following code:

```
error_log("User's email is $user->email");
```

You can add as many variables as you want this way, but they need to be strings or numbers. For other variable types such as arrays and objects, use the recursive print function, `print_r()`:

```
error_log("User object: " . print_r($user, true));
```

It can be difficult to read, but it has all the information about the user object. Just remember to set the second argument of `print_r()` to `true` so that it returns a string.

Watching the log file

1. To see if there are new lines being added to the log file, use the tail command on Linux or Macs: `tail -f error_log`.

2. Windows users need to download and install software to do this. This web page has information on possible Windows tools: `http://stackoverflow.com/questions/113121`.

To easily turn logging on and off, use Elgg's `elgg_log()` function rather than `error_log()`. It takes two parameters: the log message string and the trace level for this message.

```
elgg_log("Completing initialization of plugin', 'WARNING');
```

If the trace level of Elgg is set to 'WARNING' or lower, then the message is logged.

Debugging to the screen

There are times when it is convenient to send debugging information to the web browser. We already turned on the display of fatal errors on the **Developer Settings** page so those errors are sent to the browser. If you haven't seen one of those errors yet, then try adding the following line of code to your `world.php`:

```
echo $bad->getGUID();
```

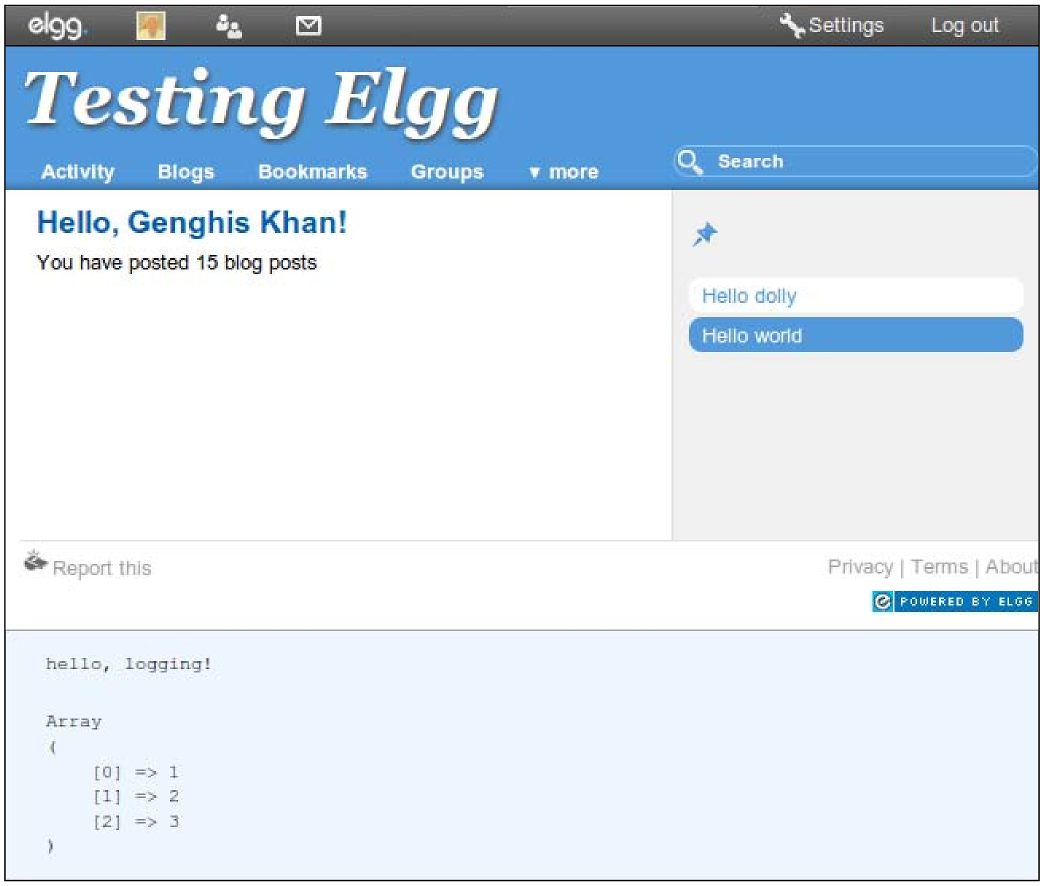
It will spit out an error message such as: *Fatal error: Call to a member function getGUID() on a non-object in /var/www/mod/hello_world/world.php on line 34*. A best practice when calling a method on an object like we did in the preceding section is to test whether the object exists:

```
if ($user) {
  echo $user->getGUID();
}
```

Logging information is sent to the web browser by using Elgg's `elgg_dump()` function and enabling the **Log to screen** option on the **Developers Settings** page. To test this, add the following code to `world.php`:

```
elgg_dump("hello, logging!");
$test = array(1,2,3);
elgg_dump($test);
```

This should be displayed on the web page at the bottom of the screen, as shown in the following screenshot:



Hello, Genghis Khan!

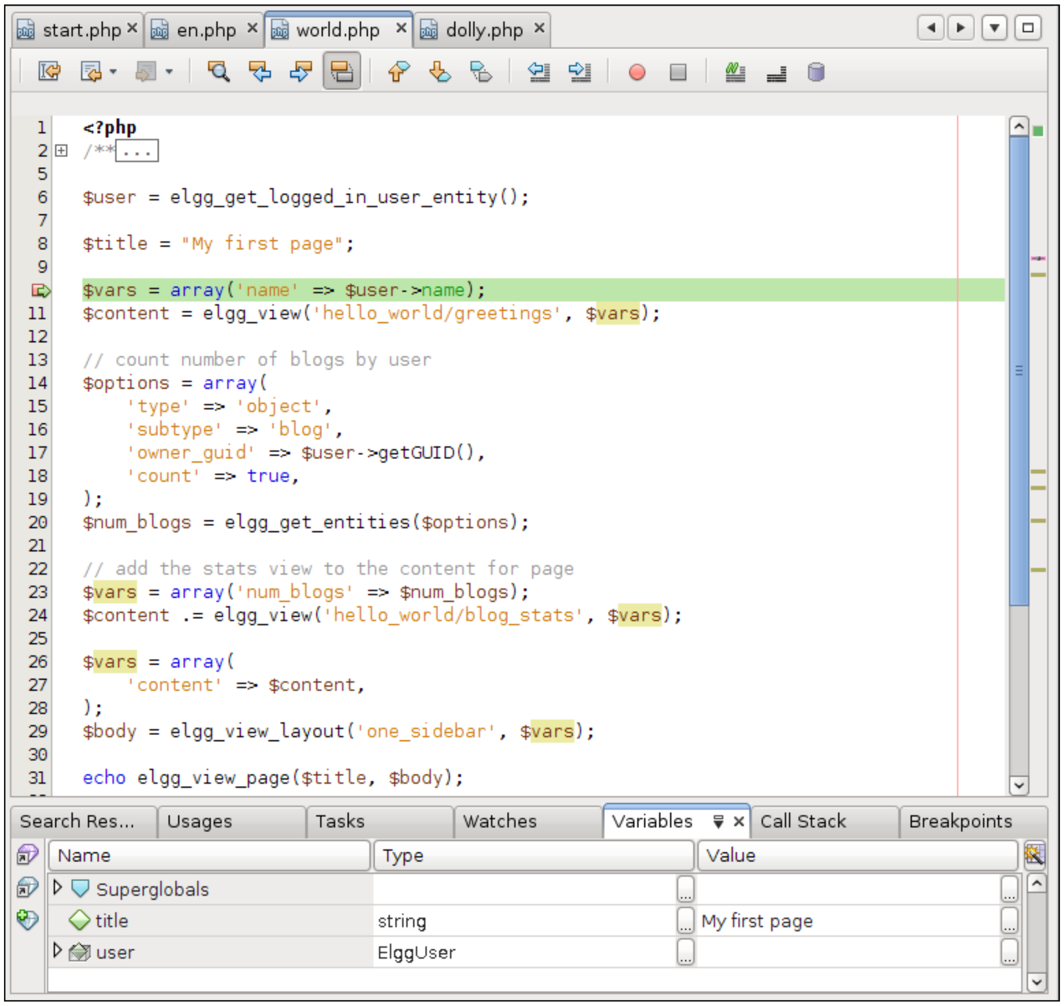
You have posted 15 blog posts

```
hello, logging!

Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

Debugging PHP through an IDE

The most powerful tool for debugging is an IDE's debugger. With a debugger, you can set breakpoints in the code, step through the code line-by-line, inspect variables at different points in the code, and look through the call stack to see how a function was called.

To use a debugger, a PHP extension must be installed on the server. Xdebug is the most popular debugging extension. There are instructions for installing it on its documentation page (`http://xdebug.org/docs/`). The PHP extension handles communication between the server and the debugger running in the IDE. Once the extension is installed and configured, the project in the IDE must be configured to talk to the server.



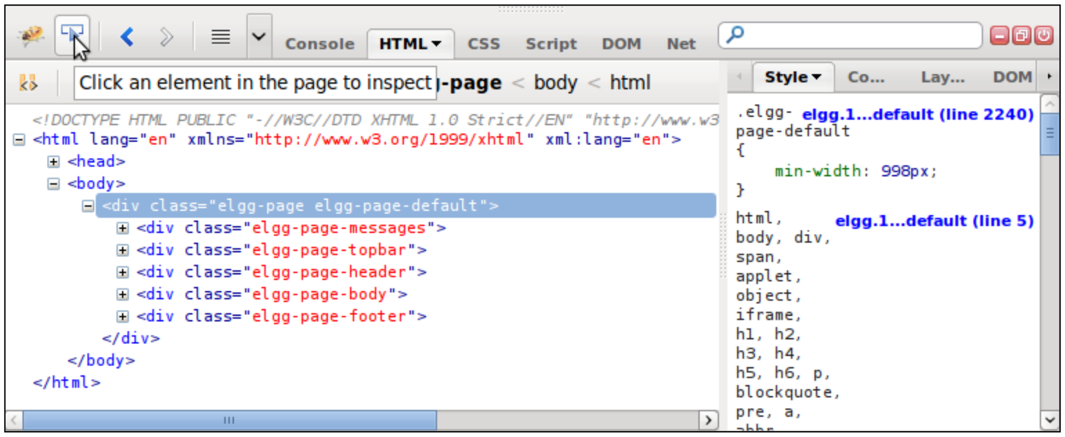
```php
<?php
/**...

$user = elgg_get_logged_in_user_entity();

$title = "My first page";

$vars = array('name' => $user->name);
$content = elgg_view('hello_world/greetings', $vars);

// count number of blogs by user
$options = array(
    'type' => 'object',
    'subtype' => 'blog',
    'owner_guid' => $user->getGUID(),
    'count' => true,
);
$num_blogs = elgg_get_entities($options);

// add the stats view to the content for page
$vars = array('num_blogs' => $num_blogs);
$content .= elgg_view('hello_world/blog_stats', $vars);

$vars = array(
    'content' => $content,
);
$body = elgg_view_layout('one_sidebar', $vars);

echo elgg_view_page($title, $body);
```

Name	Type	Value
Superglobals		
title	string	My first page
user	ElggUser	

Firebug and other browser development tools

There are tools available for every major browser for debugging HTML, CSS, and JavaScript. One of the first was Firebug (`http://getfirebug.com/`). It is an add-on for Mozilla Firefox and has become the standard that other web browser development tools are judged by. With it you can inspect and modify HTML and CSS, debug Javascript, and monitor Ajax requests.

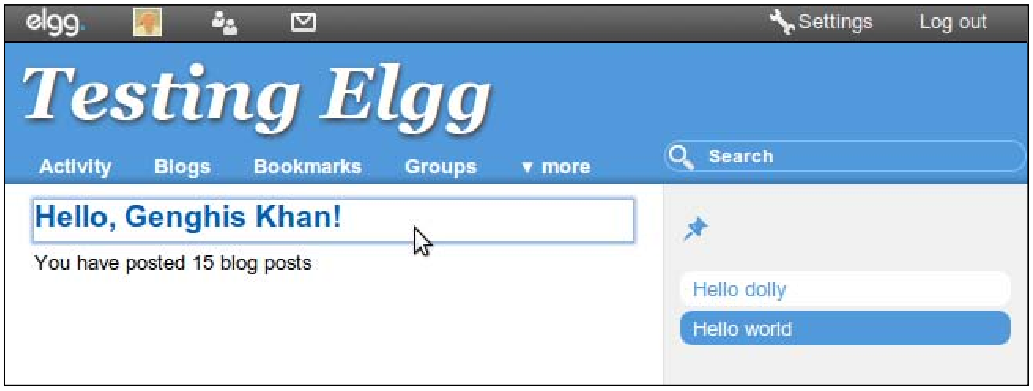
Let's modify a CSS property as an example of what is possible with Firebug. This is a common activity when perfecting a plugin's CSS. Doing this lets you quickly experiment without having to edit a file and reload a page for every change. Select the inspect tool next to the bug icon:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
  <body>
    <div class="elgg-page elgg-page-default">
      <div class="elgg-page-messages">
      <div class="elgg-page-topbar">
      <div class="elgg-page-header">
      <div class="elgg-page-body">
      <div class="elgg-page-footer">
    </div>
  </body>
</html>
```

```
.elgg-      elgg.1...default (line 2240)
page-default
{
    min-width: 998px;
}
html,       elgg.1...default (line 5)
body, div,
span,
applet,
object,
iframe,
h1, h2,
h3, h4,
h5, h6, p,
blockquote,
pre, a,
```

As you move the mouse cursor over the page, Firebug outlines the HTML elements. Mouse over the content area title and select it.

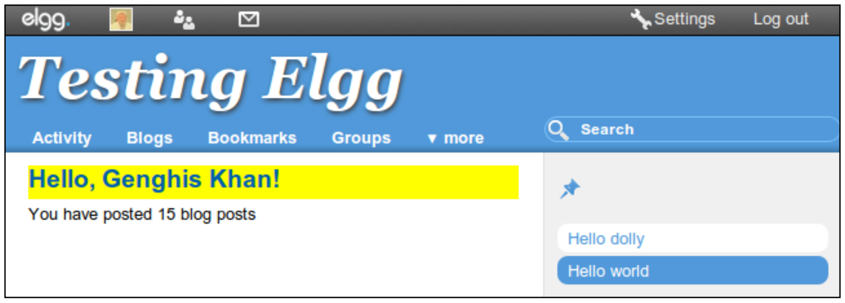


Hello, Genghis Khan!

You have posted 15 blog posts

After it is selected, the CSS that controls its appearance is shown in the right Firebug window and its HTML is highlighted in the left. Either the HTML or the CSS can be edited to try out a bug fix. This is much easier than repeatedly editing the original file and reloading the web page in your browser. To add a CSS property to the content area title, right-click on the **h2** selector in the right Style panel and select **New Property**. Add the following property to change the background of the title to yellow:

```
background-color: yellow;
```



Hello, Genghis Khan!

You have posted 15 blog posts

The more you use a tool like this, the faster your debugging will be on the client side. For other browsers, do a search on the browser's name plus *web developer tool*. There is an abundance of information about these tools on the web.

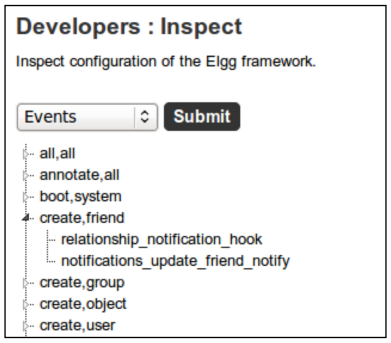
Elgg developer tools

We have already used the **Developer Settings** page, which is provided by the developer tools plugin. Two other capabilities provided by this plugin are inspecting Elgg's configuration and previewing theming elements.

Inspect

The inspect tool enabled developers to explore Elgg's configuration. In the `Hello, World` plugin, we registered for the `init, system event`, but what if we had misspelled *system*? Our handler function would not have been called and we would be wondering why the plugin was not working. The inspect tool allows us to confirm that we properly registered for an event or that we are using the correct view name with `elgg_view()`.

To use the tool, a developer selects the desired type of information and clicks submit. The tool displays that information in a tree that can be explored. In the following we are checking what functions have been registered for the `create,friend event`:

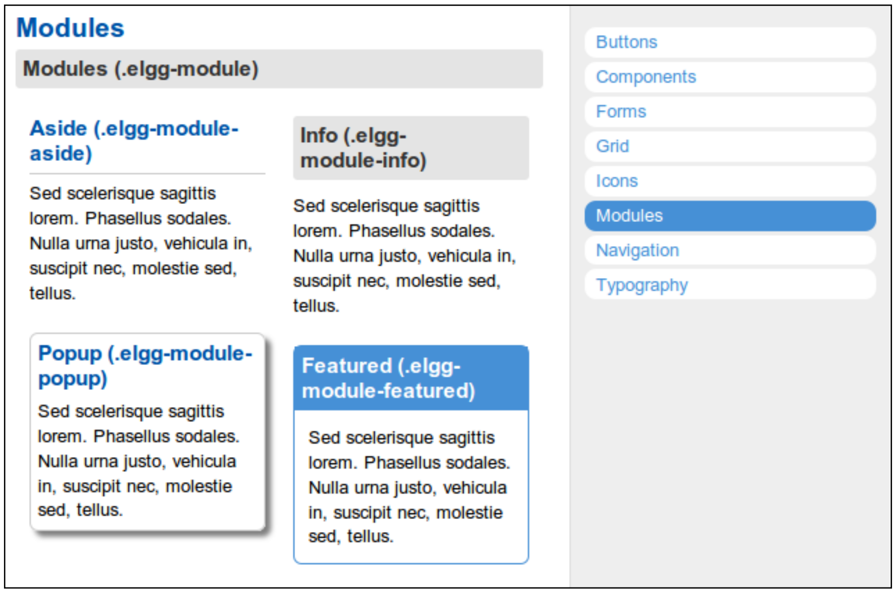


Inspect configuration of the Elgg framework.

```
all,all
annotate,all
boot,system
create,friend
    relationship_notification_hook
    notifications_update_friend_notify
create,group
create,object
create,user
```

Theming sandbox

The theming sandbox lets developers and designers view what elements are available for building and styling web pages. Each of its pages shows an element that can be used on a page with information about the CSS markup being used from Elgg's CSS framework. It is a work in progress, but in the future it should include further view information and a method for viewing the HTML source of each component.



Sed scelerisque sagittis lorem. Phasellus sodales. Nulla urna justo, vehicula in, suscipit nec, molestie sed, tellus.

Summary

This chapter guided you through the creation of your first plugin. Along the way we covered how to create views, register for events, and use the page handling system. We also included some advice on debugging plugins. All of this prepares you to continue the learning process in the next chapter, where we create nine different plugins that each demonstrates a new capability of Elgg.

8 Customization through Plugins

The previous chapter introduced new concepts by incrementally adding to a single plugin. Along the way, it covered plugin initialization, language files, creating views, page handling, the event system, and adding sidebar menus.

This chapter continues the learning process through a series of nine lessons, each with its own plugin. Every lesson explains a new aspect of Elgg that you will need to know to customize your site. The lessons define a problem that you are trying to solve, offer a general solution, and then demonstrate a specific solution through the creation of a plugin. The last section of each lesson offers suggestions on extending that plugin or provides ideas on writing a new plugin that reinforces what you have just learned.

The lessons are ordered by increasing complexity. Some of them will build off previous lessons, but all of them assume that you have worked through the `Hello, World` plugin from the previous chapter. Be sure to test the plugins often. There are particular points where testing is recommended, but do not let that stop you from testing at other points in the lesson.

The code for all the examples can be downloaded from `http://www.packtpub.com/elgg-18-social-networking/book`. The code contains additional comments from what is shown in the code snippets. There is also additional error checking that was left out of the lessons for the sake of clarity.

You will get the most out of the lessons if you work through them step-by-step. If you prefer not to type in the code listed in this chapter, then copy chunks of code from the downloaded plugins into your plugins as you work through each step. Try changing portions of the code or commenting out sections. Breaking code and then putting it back together is a great way to learn. As you figure out how things work, add comments in the code to help you remember in future.

This chapter covers the following:

- Overriding language strings
- Overriding a view
- Extending a view
- Using Elgg's event system
- Creating a widget
- Adding user settings
- Adding JavaScript
- Using a plugin hook
- Adding a plugin setting for administrators
- Working with forms and saving data

Lesson 1: Changing wording

This lesson expands your knowledge of Elgg's language system.

Problem

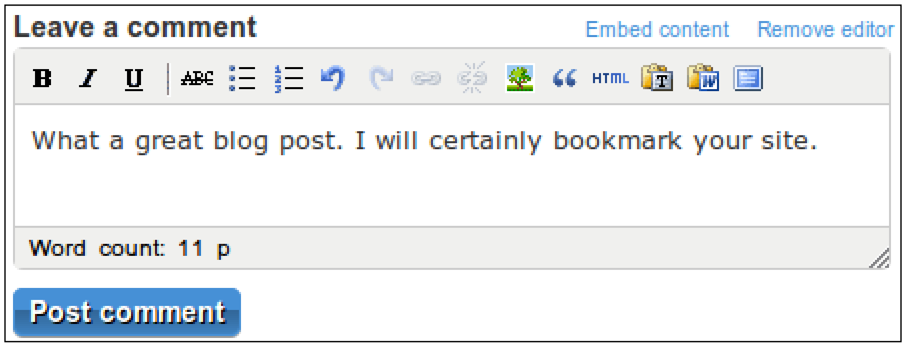
There is wording that you want to change. Elgg says `'comments'`, but you prefer `'responses'`. Elgg says `'Register'`, but you prefer `'Sign up'`. You may think that those changes involve editing lots of files, but the solution is much easier than that.

Solution

Remember the `elgg_echo()` function from last chapter that mapped strings such as `'hello:world'` to the proper phrase in the user's language ("*Hello, World!*" in English, "*¡Hola, Mundo!*" in Spanish, and so on)? Besides providing translations, we can also take advantage of `elgg_echo()` to easily change wording. We do this by overriding the language mapping using a plugin.

Example

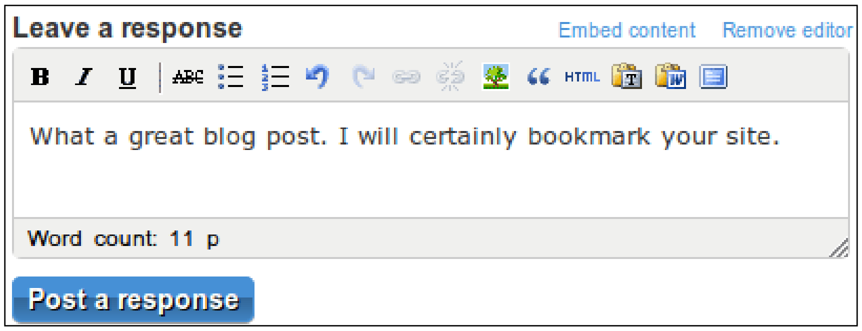
In this example, we will change the `'comment'` language to `'response'` language so that instead of displaying **Post comment** after a blog post like in the following screenshot:



What a great blog post. I will certainly bookmark your site.

Word count: 11 p

It will display **Post a response** like the following screenshot:

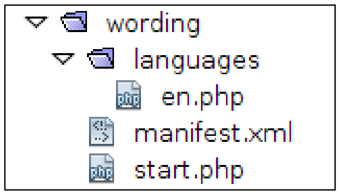


What a great blog post. I will certainly bookmark your site.

Word count: 11 p

Step 1: Create the plugin structure

Just as we did with the `Hello, World` plugin, create a directory for the plugin in `/mod` called **wording** and add the files **manifest.xml** and **start.php**. As we are working with the language system of Elgg, add the **languages** directory and the **en.php** file. The plugin structure should look like the following image:



Update the manifest file to reflect the purpose of this plugin. The **start.php** file will be left empty (we include it so that Elgg knows that this is a proper plugin.) All of our work will be in the **en.php** language file.

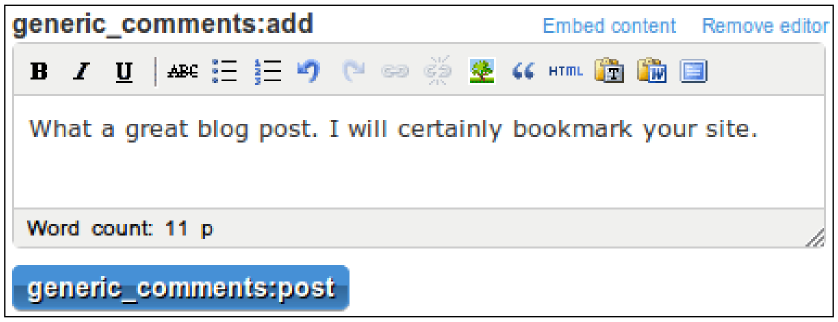
Step 2: Find the language strings

We have two options for finding the language strings that Elgg uses for the comment text:

- Use the editor to do a search through all the PHP files in Elgg looking for the string `"Post comment"`. This approach finds the mapping in `/languages/en.php` (the primary language file of Elgg):

  ```
  'generic_comments:add' => "Leave a comment",
  'generic_comments:post' => "Post comment",
  'generic_comments:text' => "Comment",
  'generic_comments:latest' => "Latest comments",
  ```

- Go to the **Developers Settings** page and enable the option for viewing the raw translation strings. This option effectively turns off `elgg_echo()` so that the comment box looks like the following screenshot:


generic_comments:add

What a great blog post. I will certainly bookmark your site.

Word count: 11 p

Step 3: Override the language string

Open the plugin's language file, `en.php`, and define the language mapping:

```
$english = array(
  'generic_comments:post' => "Post a response",
  'generic_comments:text' => "Response",
);

add_translation("en", $english);
```

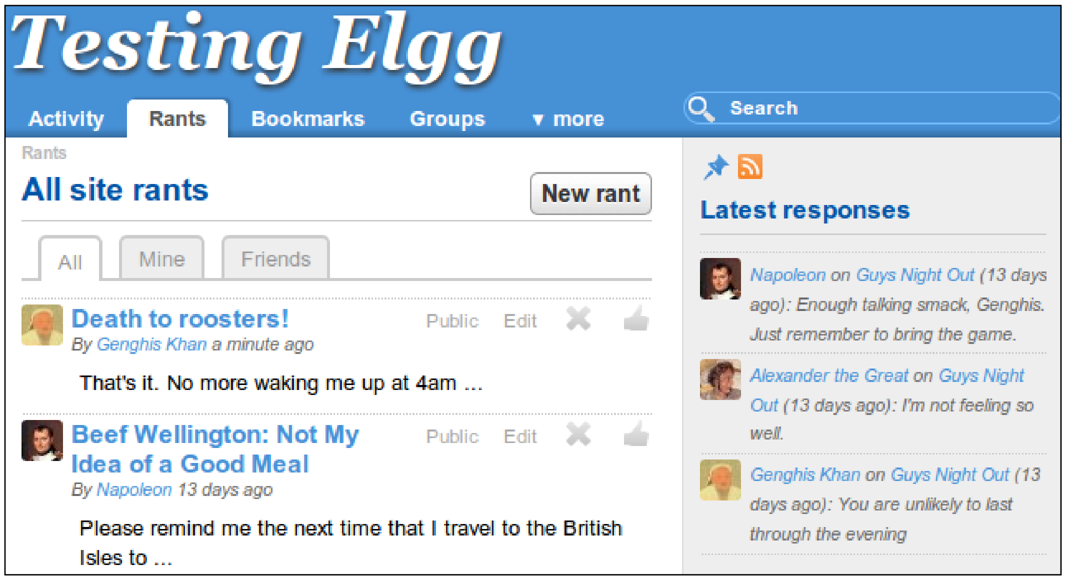
Copy the plugin onto the server and activate it. When you view a page with a comment box, you will see the response language instead of comment language. Completing this task requires searching for any other use of the word comment in the language files (both core and plugins) and adding them to this plugin's language file.

Plugin order matters

To modify the language strings used in a plugin, your plugin must be loaded after that plugin. This means your plugin must be listed after the plugin being modified on the `Plugin` page.

Exercise

To practice this further, change all the language for the blog plugin so that instead of posting a blog, users post rants. Try using both the technique of searching through files and using the language option on the **Developer Settings** page. When you are done, you should see something like the following screenshot:



Death to roosters!
By Genghis Khan a minute ago
That's it. No more waking me up at 4am ...

Beef Wellington: Not My Idea of a Good Meal
By Napoleon 13 days ago
Please remind me the next time that I travel to the British Isles to ...

Napoleon on Guys Night Out (13 days ago): Enough talking smack, Genghis. Just remember to bring the game.

Alexander the Great on Guys Night Out (13 days ago): I'm not feeling so well.

Genghis Khan on Guys Night Out (13 days ago): You are unlikely to last through the evening

Lesson 2: Modifying a section of a page

Overriding a view is covered in this lesson.

Problem

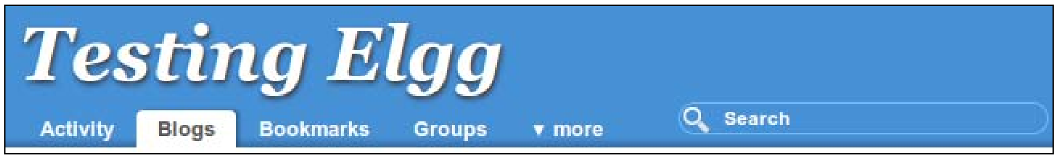
There is a section of the Elgg page structure that you want to change. Perhaps you want to include an ad in the footer or rearrange the topbar. You could search for that specific HTML code in Elgg and hack it to get what you want, but then upgrades to the latest version of Elgg would be difficult. You want to do this the right way.

Solution

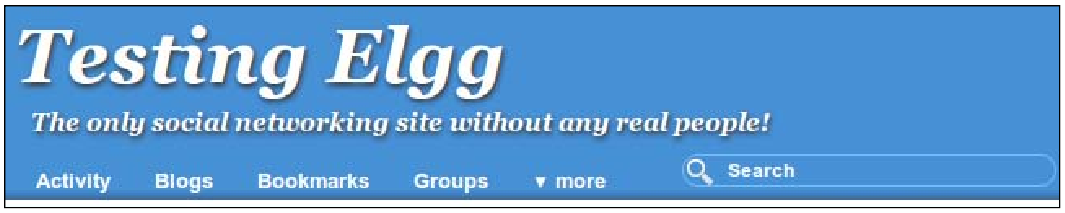
Elgg divides up its page structure into views. As we discussed in the last chapter, a view is a chunk of HTML code. The topbar and header are views. The search box is a view inside the header view. By changing one of these views, we change every Elgg page that uses it. With Elgg, the primary way to change a view is to override it. Overriding a view results in our HTML replacing the HTML of the core view.

Example

The default Elgg theme has a header with the site name, a search bar, and a set of menu tabs:



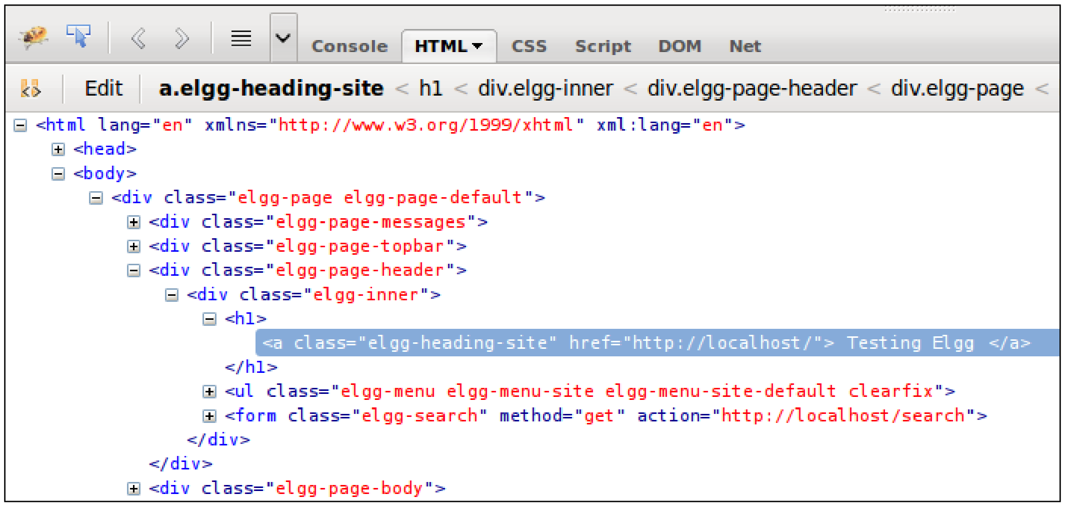
We are adding a tagline below the site name, as shown in the following screenshot:



Step 1: Find the view to override

All the HTML views for the core Elgg engine are stored in `/views/default`. Looking through all the directories could take a lot of time, so we will try two different approaches to find the view with the site name in it:

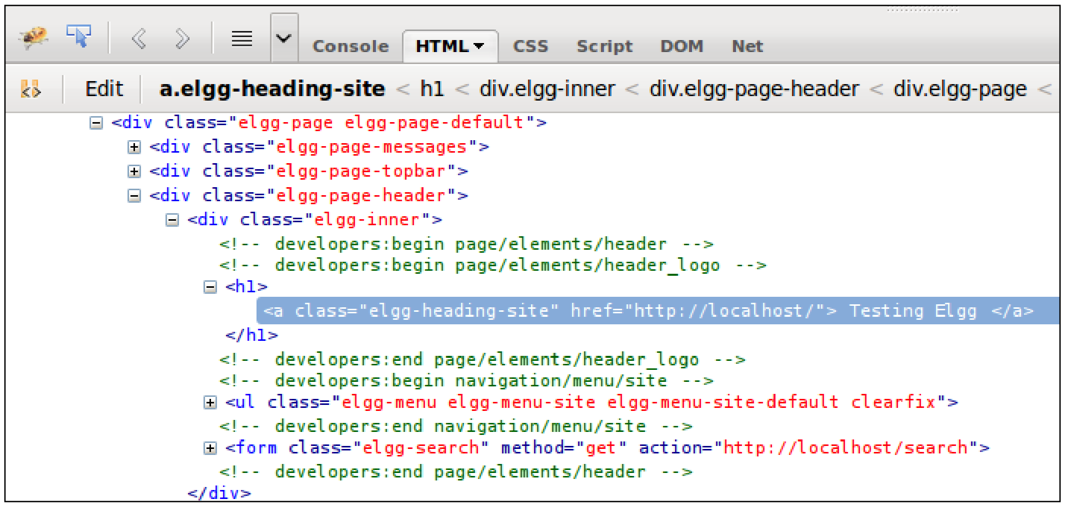
1. Using Firebug or the equivalent tool for your browser, start the inspect tool and click on the site title in the header. The Firebug panel will display the HTML code for that area of the page. You will notice a link with a class of `elgg-heading-site`:


Console
HTML
CSS
Script
DOM
Net
Edit
a.elgg-heading-site < h1 < div.elgg-inner < div.elgg-page-header < div.elgg-page <

```
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
  <body>
    <div class="elgg-page elgg-page-default">
      <div class="elgg-page-messages">
      <div class="elgg-page-topbar">
      <div class="elgg-page-header">
        <div class="elgg-inner">
          <h1>
            <a class="elgg-heading-site" href="http://localhost/"> Testing Elgg </a>
          </h1>
          <ul class="elgg-menu elgg-menu-site elgg-menu-site-default clearfix">
          <form class="elgg-search" method="get" action="http://localhost/search">
        </div>
      </div>
      <div class="elgg-page-body">
```

2. Searching for the string `elgg-heading-site` should find the view that we want to override.

Another option is using the **Developer Settings** page. It has a setting called `Wrap views`. When it is enabled, comments are placed around each view with the name of the view in the comment. By configuring Firebug to show comments, we can use the inspect tool to select the site name and find that it is wrapped by the comment for the `'page/elements/header_logo'` view.


Console
HTML
CSS
Script
DOM
Net
Edit
a.elgg-heading-site < h1 < div.elgg-inner < div.elgg-page-header < div.elgg-page <

```
<div class="elgg-page elgg-page-default">
  <div class="elgg-page-messages">
  <div class="elgg-page-topbar">
  <div class="elgg-page-header">
    <div class="elgg-inner">
       <!-- developers:begin page/elements/header -->
       <!-- developers:begin page/elements/header_logo -->
      <h1>
        <a class="elgg-heading-site" href="http://localhost/"> Testing Elgg </a>
      </h1>
       <!-- developers:end page/elements/header_logo -->
       <!-- developers:begin navigation/menu/site -->
      <ul class="elgg-menu elgg-menu-site elgg-menu-site-default clearfix">
       <!-- developers:end navigation/menu/site -->
      <form class="elgg-search" method="get" action="http://localhost/search">
       <!-- developers:end page/elements/header -->
    </div>
```

We now know that the view is located at `/views/default/page/elements/header_logo.php`.

Step 2: Create the plugin structure

We call our plugin **custom_logo**. It needs a plugin manifest file and `start.php`. Add those files and update the manifest to describe the purpose of the plugin.

Overriding a view requires that we add a view in the same relative location in our plugin as it exists in the default Elgg theme. As the view is located at:

`/views/default/page/elements/header_logo.php`

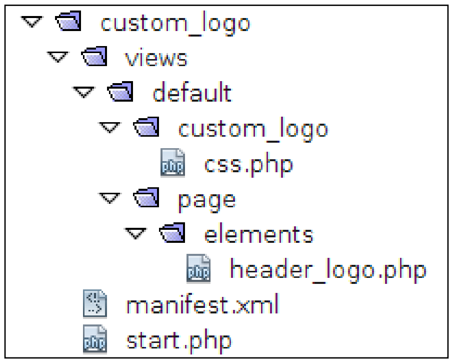
We create our view at:

`/mod/custom_logo/views/default/page/elements/header_logo.php`.

When Elgg detects the view there, it automatically replaces the core view with our view.

The last file our plugin needs is a CSS view so that we can style our tagline. Create a file named `css.php` in a `custom_logo` directory under `views/default/`. The reason that we put our plugin's CSS view in a directory named after the plugin is to prevent conflicts with other plugins. This is a best practice for plugin development.

After the files and directories have been created, we have the following plugin structure:



```
custom_logo
  views
    default
      custom_logo
        css.php
      page
        elements
          header_logo.php
  manifest.xml
  start.php
```

Step 3: Edit the logo view

We could start with a blank view or we could copy the current view code into our view as a starting point. Let's copy the code as we only want to add the tagline. After we do that, our view's code looks like the following:

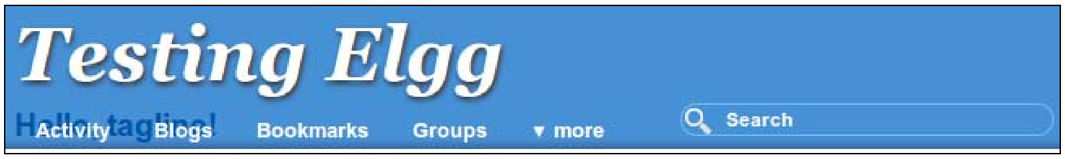
```
<?php
/**
 * Elgg header logo
 */

$site = elgg_get_site_entity();
$site_name = $site->name;
$site_url = elgg_get_site_url();
?>

<h1>
  <a class="elgg-heading-site" href="<?php echo $site_url; ?>">
    <?php echo $site_name; ?>
  </a>
</h1>
```

Now is a great time to make a small edit, copy the plugin to our server, and make sure we are changing the correct view. Remember that it is a good idea to test your plugins often as you write them. Insert some arbitrary text directly above the `<h1>` element and then check that it appears in the header when you view the website.

To add a tagline, we append our text to the bottom of the view and wrap it in a heading tag:

```
<?php
/**
 * Overriding the page/elements/header_logo view
 */

$site = elgg_get_site_entity();
$site_name = $site->name;
$site_url = elgg_get_site_url();
?>

<h1>
  <a class="elgg-heading-site" href="<?php echo $site_url; ?>">
    <?php echo $site_name; ?>
  </a>
</h1>
<h2 class="custom-logo-heading">
  Hello, tagline!
</h2>
```

The tagline appears in the header, but is extremely difficult to read due to its position and color. We need to style it with CSS, as follows:



Step 4: Style the header

Before we add any CSS to the `'custom_logo/css'` view, we need to inform Elgg that we want that view included with the main CSS file. We do this in our `start.php` file, as follows:

```
elgg_register_event_handler('init', 'system', 'custom_logo_init');

function custom_logo_init() {
  elgg_extend_view('css/elgg', 'custom_logo/css');
}
```

This code tells Elgg to add the output of `'custom_logo/css'` to that of `'css/elgg'`. If we wanted to add CSS to the admin theme instead, then we would extend the `'css/admin'` view.

Remember to turn off caching

Whenever you are writing new plugins or themes, turn off caching. Otherwise, your changes may not show up.

In our plugin's `css.php` file, we add the following:

```
.elgg-page-default .elgg-page-header > .elgg-inner {
  height: 120px;
}
.custom-logo-heading {
  color: white;
  font-family: Georgia,times,serif;
  font-style: italic;
  font-size: 1.2em;
  text-shadow: 1px 2px 4px #333333;
}
```

The first statement makes the header taller so that our tagline will not overlap with the site menu. The second part styles the tagline text giving it the appearance of white text set off by a dark shadow. When you view your site now, it should look like the screenshot from the beginning of this lesson.

Exercise

You can continue to work with the `custom_logo` by replacing the site name heading with an image. You can also pull the tagline text from the optional site description that is set on the **Basic Settings** page in the administration area. The text is retrieved like the following:

```
<h2 class="custom-logo-heading">
  <?php echo $site->description; ?>
</h2>
```

To practice overriding views, try replacing the footer of Elgg with your own custom footer. After overriding views a few times, the process of finding the view and setting up the plugin will become easy for you. Then, you can focus on writing the HTML code and the CSS.

If you override a view that belongs to a plugin, then your plugin must be loaded after that plugin.

Lesson 3: Adding new content to a page

Extending a view is demonstrated in this lesson.

Problem

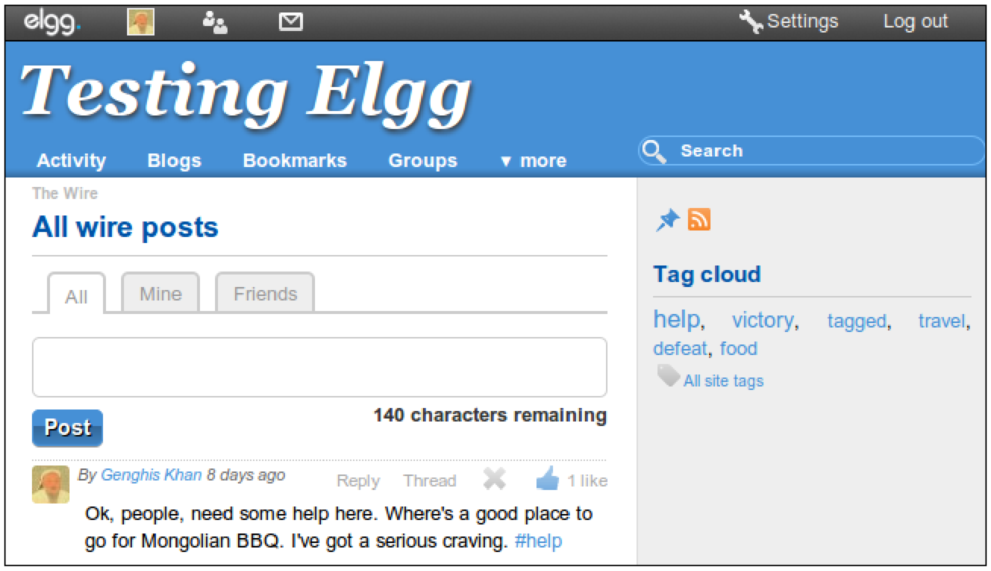
Instead of modifying an area of a page as in the preceding lesson, you want to add new content. This could be including a "`Share this`" link under every blog, a welcome message to visitors under the header, or a Twitter feed in the sidebar. What is the best way to do this with Elgg?

Solution

Instead of overriding a view, we can extend it. Extending a view adds our content directly below (or above) the view. This is the easiest way to insert new content into a page.

Example

In our example, we would like to add some tips on how to use the site on each page. A good location for these tips is under the sidebar menu. Currently the sidebar looks like the following screenshot:



All wire posts

140 characters remaining

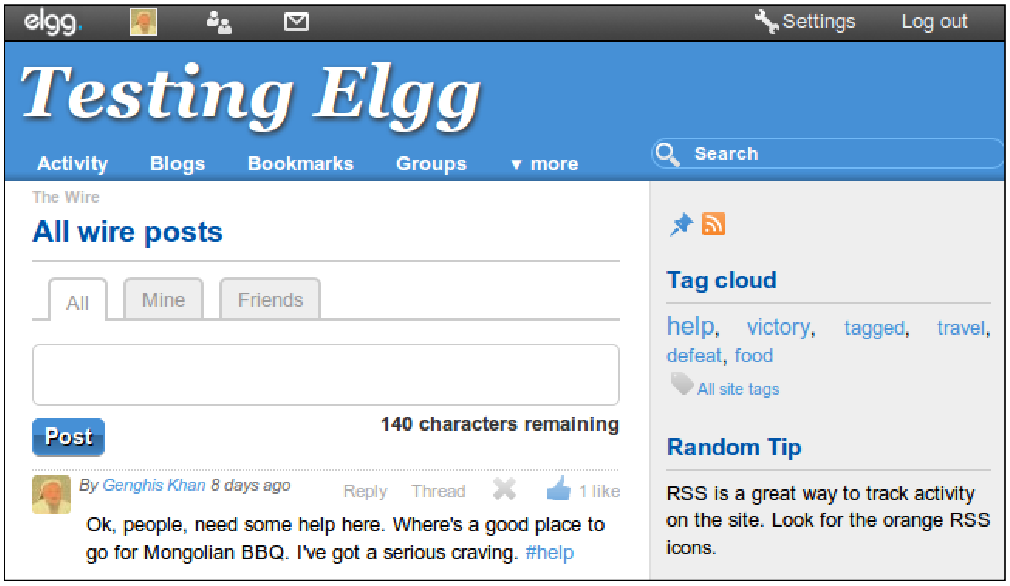
By Genghis Khan 8 days ago

Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

Tag cloud

help, victory, tagged, travel, defeat, food

We want to add our helpful tip to the sidebar as shown in the following screenshot:



All wire posts

140 characters remaining

By Genghis Khan 8 days ago

Ok, people, need some help here. Where's a good place to go for Mongolian BBQ. I've got a serious craving. #help

Tag cloud

help, victory, tagged, travel, defeat, food

Random Tip

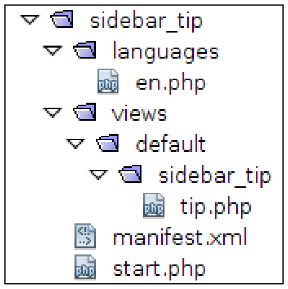
RSS is a great way to track activity on the site. Look for the orange RSS icons.

Step 1: Find the view to extend

The quickest way to find the view to extend is using the **Wrap views** option on the **Developer Settings** page to wrap all the views with comments as we did in the preceding lesson. Using Firebug to inspect the HTML, we find that there are many views due to the navigation system. Just inside the `div` with the class `elgg-sidebar` is a comment for the `'page/elements/sidebar'` view. That is the view that we will extend.

Step 2: Create the plugin structure

We call our plugin **sidebar_tip**. It needs a plugin manifest file, **start.php**, a language file, and our view. We name the view `'sidebar_tip/tip'`. Notice how each time we create a new view, we place it in a directory named after our plugin. Doing this ensures that another plugin will not accidentally override our view. The structure looks like the following:



```
sidebar_tip
  languages
    en.php
  views
    default
      sidebar_tip
        tip.php
    manifest.xml
    start.php
```

Step 3: Build our view

In our view, we need a heading and the text of the tip. We want our tip area to look like the other sidebar boxes in Elgg. If you remember from the previous chapter, there was a page on the `Theming Preview` for modules. Modules are blocks of content with a header and a body. Elgg makes it easy to create modules using the function `elgg_view_module()`. Using Firebug again, we can check what type of module is used for the `tagcloud`. The CSS classes for that module are `.elgg-module` and `.elgg-module-aside`. This means the module type is "`aside`". Our view with a hard-coded tip looks like the following:

```
<?php
/**
 * Tip view - it is added to the sidebar
 */

$title = elgg_echo('sidebar_tip:title');
$text = "This is my test tip";
echo elgg_view_module('aside', $title, $text);
```

We need the `echo` on the last line or our sidebar tip will not be sent to the browser. Not adding an echo before an `elgg_view function` is a common mistake for both Elgg veterans and rookies.

Before testing the plugin, define the language string '`sidebar_tip:title`' in our en.php file and extend the '`elgg_sidebar/extend`' view in our plugin's initialization function.

```
elgg_register_event_handler('init', 'system', 'sidebar_tip_init');

function sidebar_tip_init() {
  // Add our sidebar_tip/tip view to the sidebar
  elgg_extend_view('page/elements/sidebar', 'sidebar_tip/tip');
}
```

The plugin is now ready for its first test. In your browser, visit a page with a sidebar and you should see the test tip that you created.

Step 4: Make the tips random

We want to rotate through a series of tips and have the order be random. We do this by defining the tips in the language file and then adding a function in `start.php` that returns a random tip. The language file looks like the following:

```
<?php

$english = array(
  'sidebar_tip:title' => 'Random Tip',

  'sidebar_tip:tip1' => 'Never spit into the wind.',
  'sidebar_tip:tip2' => 'You can set your notification options on
    the settings page.',
  'sidebar_tip:tip3' => "See something that shouldn't be here?
    Report it using the whistle icon.",
  'sidebar_tip:tip4' => "RSS is a great way to track activity on
    the site. Look for the orange RSS icons.",
);

add_translation("en", $english);
```

The function that returns the random tip creates a string starting with `'sidebar_tip:tip'` and appends a number. That string is fed into `elgg_echo()` and out comes our tip text.

```
function sidebar_tip_get_tip() {
  $num_tips = 4;

  $select = rand(1, $num_tips);
  return elgg_echo("sidebar_tip:tip$select");
}
```

> If you are not familiar with the function rand(), then you can look up its documentation by going to `http://www.php.net/rand`. In fact, any PHP function can be looked up by visiting `http://www.php.net/<function name>`.

To finish the plugin, we replace the hard-coded tip in the view with the following:

```
$text = sidebar_tip_get_tip();
```

Each time we load a page, Elgg includes a different tip.

Exercise

To extend this example, determine if the viewer is logged in and display a message encouraging visitors to join the site.

```
if (elgg_is_logged_in()) {
  $title = elgg_echo('sidebar_tip:title');
  $text = sidebar_tip_get_tip()
} else {
  // alternate content here for visitors
}
echo elgg_view_module('aside', $title, $text);
```

To practice extending views, try extending `'page/element/header'` so that you can post announcements to your users.

> **Adding content before a view**
> Sometimes you may want to add content above a view rather than below it. You can do that by calling `elgg_extend_view()` and including a third parameter which sets its priority. Any number less than 500 will cause your view to be added to the page before the primary view.

Lesson 4: Doing something when X happens

This lesson explains how to use Elgg's event system.

Problem

Whenever someone joins a group, you want to send an e-mail welcoming them. Whenever a blog entry is posted, you want to be notified so that you can review its content. Whenever someone logs in, you want to increase a counter so that you can proudly post on your front page how many times people have used your site: "*3 billion served*". In all of these cases, an event occurs and you want something to happen.

Solution

Elgg has an event system that calls a function when something occurs. When a user logs in, a login event is triggered within Elgg. Events are triggered for updating profile fields, joining a group, forming a friendship, uploading a file, or leaving a comment. When an event occurs, Elgg calls the functions that are registered for that event. Each function runs and when it is done, control is returned to Elgg.

Example

We want to receive an e-mail whenever someone joins our site. Our site is new and we want to check that new members are able to validate their e-mail addresses and give them a personal greeting.

Step 1: Find the event

We have three options for finding an event to use for our new user notification plugin, as follows:

1. Check the Elgg wiki at `http://docs.elgg.org/List_of_Events`. The developers FAQ includes a list of the events and a description of when the events occur.
2. Search through the code looking for the string "`elgg_trigger_event`". This locates every place in the code where an event occurs. Then, you can narrow the possibilities based on the names of the events.

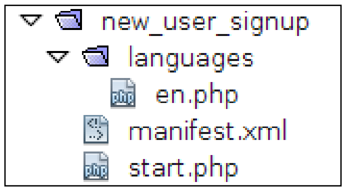
3. Enable logging of events and plugin hooks on the **Developers Settings** page and turn off logging to the screen. For this example, register a test user and then look through the events that were logged. Each event description in the log includes the event's name and the name of the function that triggered it. One section of the log should look like the following:

```
Plugin hook: 'registeruser:validate:password, all' in
  validate_password()
Plugin hook: 'registeruser:validate:username, all' in
  validate_username()
Plugin hook: 'container_permissions_check, user' in
  create_entity()
Event: 'create, member_of_site' in add_entity_relationship()
Event: 'create, user' in create_user_entity()
Event: 'create, metadata' in create_metadata()
```

Among the different choices, the `create/user` event looks like the best choice. Each event is typically described by an action word (`create`) and a noun (`user`).

Step 2: Create the plugin structure

This plugin is called `new_user_signup`. We need a plugin manifest, a start script, and a language file. The plugin structure looks like the following:



We register for the `'create, user'` event in the plugin's initialization function, as follows:

```
elgg_register_event_handler('init', 'system',
  'new_user_signup_init');

function new_user_signup_init() {
  elgg_register_event_handler('create', 'user',
    'new_user_signup_notify');
}
```

We define the `new_user_signup_notify()` function in the next step.

Step 3: Write the function that sends the e-mail

Our function that runs when the `'create, user'` event is triggered calls `elgg_send_email()` to send us an e-mail about the new user. For now, we hard-code the e-mail address and then create the subject and message body using `elgg_echo()`.

```
function new_user_signup_notify($event, $type, $user) {

  $to = "me@example.org";

  // get site email address for the from address
  $site = elgg_get_site_entity();
  $from = $site->email;

  $name = $user->name;
  $username = $user->username;

  $subject = elgg_echo('new_user:subject', array($site->name));
  $message = elgg_echo('new_user:body', array($name, $username));

  elgg_send_email($from, $to, $subject, $message);
}
```

The language strings in `en.php` are defined like the following:

```
$english = array(
  'new_user:subject' => "New user signup on %s",
  'new_user:body' => "A new user just signed up with a name of %s
  and a username of %s",
);

add_translation("en", $english);
```

The message body provides all the information that we need to track down this user if there is any difficulty in the validation process. If you do not have e-mail configured for your test server, then send this message to the log instead with `elgg_log()`.

Exercise

To improve your understanding of events, you must figure out what event is triggered when a user friends another user. Register a function for that event and send out notifications to yourself.

Lesson 5: Creating a custom widget

Elgg's widget system is demonstrated through this lesson.

Problem

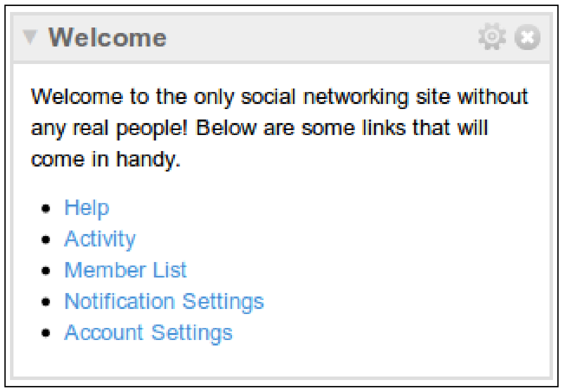
You need to add some custom content to your users' profiles. This may be information related to the purpose of the site (a site for sports fans could allow members to display the schedule for their favorite team on their profiles). It could be statistics about how many people have read their latest blog posts or looked through their newest photos. You may want to add a special welcome to new members or provide links to featured content.

Solution

Elgg has a widget framework that makes it easy to add content like that described above to profiles and dashboards. All that is required to create a new widget is adding two views and registering the widget. Elgg handles all of the details.

Example

In this example, we are adding a widget that welcomes new users and provides a list of useful links. We allow the user to set the number of links that are displayed. The widget looks like the following screenshot:

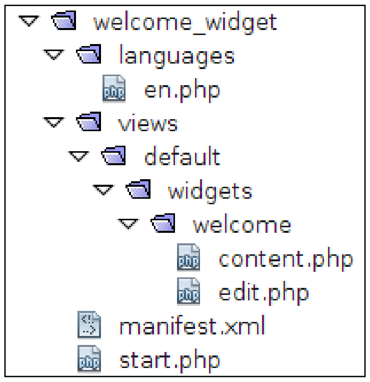


Welcome to the only social networking site without any real people! Below are some links that will come in handy.

- Help
- Activity
- Member List
- Notification Settings
- Account Settings

Step 1: Create the plugin structure

Our `welcome_widget` plugin has a plugin manifest, start script, language file, and two widget views.



We register the widget in the plugin's initialization function, as follows:

```
elgg_register_event_handler('init', 'system',
  'welcome_widget_init');

function welcome_widget_init() {
  $title = elgg_echo('welcome');
  $description = elgg_echo('welcome:widget:description');
  elgg_register_widget_type('welcome', $title, $description);
}
```

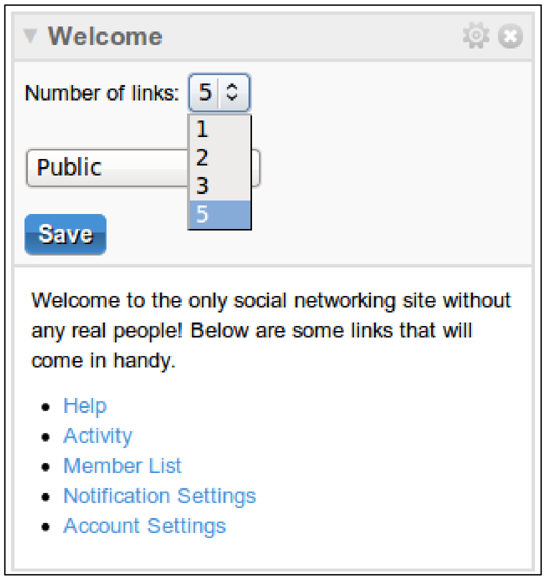
The first parameter in `elgg_register_widget_type()` is the identifier of the widget. It tells Elgg where to find the views for this widget. As we used "`welcome`" as the identifier, our widget's views are in `views/default/widgets/welcome/`. The second parameter is the title and the third is the description used for the tool tip on the add widget panel. There are also some optional parameters not used here. You can look up the parameters for functions using the API reference at `http://reference.elgg.org/` or by checking the source code.

Step 2: Create the widget edit view

Our plugin lets the user set the number of displayed links so we add an edit view (`edit.php`) with the following code:

```
<?php
/**
 * Widget edit code
 * User selects how many links to display in widget
 */

// set default value
if (!isset($vars['entity']->num_links)) {
  $vars['entity']->num_links = 5;
}

// when Elgg handles the saving, names need to be params[<var
  name>]
$params = array(
  'name' => 'params[num_links]',
  'value' => $vars['entity']->num_links,
  'options' => array(1, 2, 3, 5),
);
$dropdown = elgg_view('input/dropdown', $params);

?>
<div>
  <?php echo elgg_echo('welcome:num_links'); ?>:
  <?php echo $dropdown; ?>
</div>
```

The output of this view is shown in the following screenshot:



Welcome to the only social networking site without any real people! Below are some links that will come in handy.

- Help
- Activity
- Member List
- Notification Settings
- Account Settings

It creates a drop-down input control for selecting the number of displayed links. Elgg handles saving the user input in the database for you. The only requirement is that the name of the input fields must be `params[<variable name>]`. That is why the name of the input field is `'params[num_links]'` as the variable is accessed by `$vars['entity']->num_links`. The access control shown on the widget below is automatically added by Elgg for all profile widgets.

Step 3: Create the widget content view

The display view (`content.php`) gets the number of links from the widget object (`$vars['entity']`). It loops over the links and adds them to an unordered list.

```
<?php
/**
 * Widget display code
 */

$num_links = $vars['entity']->num_links;

// welcome text
echo elgg_echo('welcome:message');

// array of links
$links = array(
  'help',
  'activity',
  'members',
  'notifications/personal',
  'settings'
);

// display links in an unordered list
echo '<div class="elgg-output">';
echo '<ul>';
for ($index = 0; $index < $num_links; $index++) {
  $link = $links[$index];
  $text = elgg_echo("welcome:text:$link");

  // create anchor tag <a href=""></a>
  $anchor = elgg_view('output/url', array(
    'href' => $link,
    'text' => $text,
  ));
  echo "<li>$anchor</li>";
}
echo '</ul>';
echo '</div>';
```

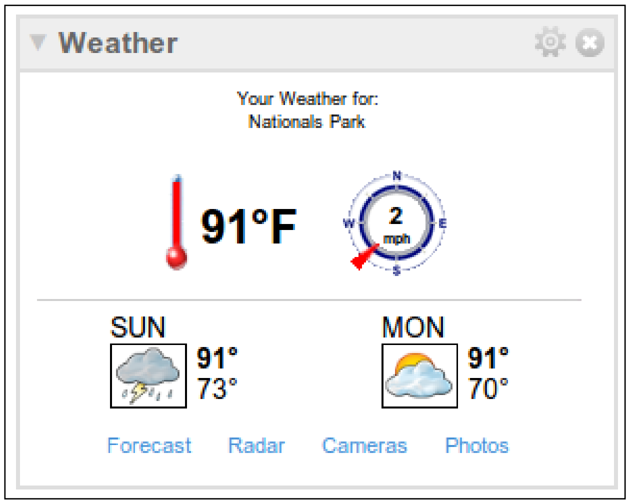
The links are pulled from the language file in a manner similar to how we selected the random tip in the `sidebar_tip` plugin. Each link address is used as a part of the language string descriptor for the text of the link.

```
<?php
/**
 * Language file for welcome widget
 */

$english = array(
  'welcome' => "Welcome",
  'welcome:widget:description' => "A welcome along with useful
    links",

  'welcome:num_links' => "Number of links",

  'welcome:message' => "Welcome to the only social networking site
    without any real people! Below are some links that will come in
    handy.",

  'welcome:text:help' => "Help",
  'welcome:text:activity' => "Activity",
  'welcome:text:members' => "Member List",
  'welcome:text:notifications/personal' => "Notification
    Settings",
  'welcome:text:settings' => "Account Settings",
);

add_translation("en", $english);
```

We did not define the entire link address so that this widget can be used on any site. Instead, we submit the partial address (such as "`settings`") to the `'output/url'` view and Elgg handles creating the full address for us (such as `http://example.org/settings`).

Exercise

Many social media sites such as Flickr, Twitter, and Digg provide snippets of JavaScript to use in widgets. Weather sites are another good source for widgets. You can copy their JavaScript code and paste it into the content view of a widget. If the site's JavaScript has a parameter, like zip code for weather widgets, then you can add an edit view so that the user can specify the parameter and then insert it into the JavaScript using a `php` function such as `str_replace()`. Here is an example of a weather widget created using code from WeatherBug (`http://weather.weatherbug.com/desktop-weather/web-widgets.html`):



Lesson 6: Giving your users options

Adding settings for users from a plugin is the topic of this lesson.

Problem

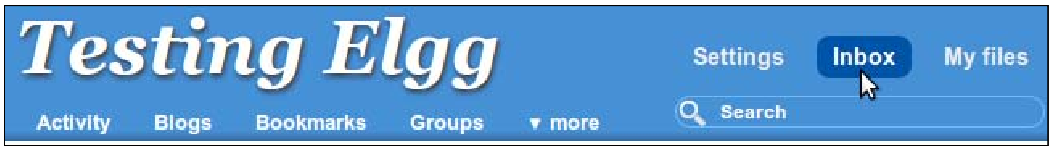
You want to give your users control over different aspects of the site. Maybe they get to pick the background color of their profile page. Or how about letting them select how comments are sorted on their blog posts – newest or oldest at the top. You might also have a plugin that they need to configure before they can use it - something like a plugin that pulls in their latest Flickr photos and imports them into a photo gallery. How do you give users these types of settings?

Solution

Just as plugins can have administrative settings, they can also have user settings. These settings are adjusted on the **Configure your tools** page found in the user's settings area. When a plugin has a view in a particular location (`views/default/plugins/<plugin_name>/usersettings.php`), Elgg automatically adds a section to this page.

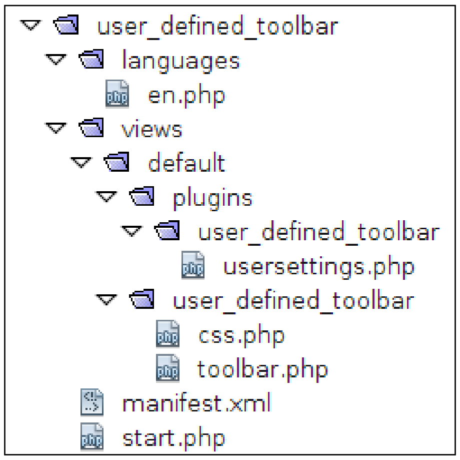
Example

We are building a toolbar that sits to the right of the site name in the header. Users control what items are shown in the toolbar through their tools settings pages, as follows:



Step 1: Create the plugin structure

The **user_defined_toolbar** has the typical plugin manifest, start script, and language file. In addition, it has three views: css, user settings, and the actual toolbar view, as follows:



```
user_defined_toolbar
    languages
        en.php
    views
        default
            plugins
                user_defined_toolbar
                    usersettings.php
            user_defined_toolbar
                css.php
                toolbar.php
    manifest.xml
    start.php
```

In `start.php`, we extend both the main CSS view and the header view.

```
elgg_register_event_handler('init', 'system',
  'user_toolbar_init');

function user_toolbar_init() {
  // add our css
  elgg_extend_view('css/elgg', 'user_defined_toolbar/css');

  // add our toolbar to the header
  elgg_extend_view('page/elements/header',
    'user_defined_toolbar/toolbar');
}
```

Step 2: Add user settings

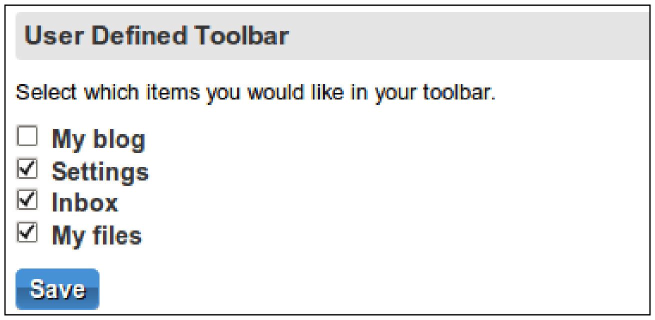
If Elgg finds a file at `/views/default/plugins/<plugin name>/usersettings.php`, then it adds a section to the **Configure your tools** page. Open the user settings view and add the following code:

```
<?php
/**
 * User settings edit code
 * user selects menu options from a list
 */

// get instructions text
$instructions = elgg_echo('udt:instruct');

// get previously saved settings
$guid = elgg_get_page_owner_guid();
$settings = elgg_get_all_plugin_user_settings($guid,
  'user_defined_toolbar');

// setup checkboxes for toolbar options
$checkboxes = '';
$tools = array('blog', 'settings', 'inbox', 'files');
foreach ($tools as $tool) {
  $label = elgg_echo("udt:$tool");
  $input = elgg_view('input/checkbox', array(
    'name' => "params[$tool]",
    'value' => 1,
    'checked' => (bool)$settings[$tool],
  ));
  $checkboxes .= "<label>$input$label</label><br />";
}

// output our user settings area
echo "<p>$instructions</p>";
echo "<div>$checkboxes</div>";
```

This view displays instructions on how to personalize the toolbar followed by checkboxes for the various options. It retrieves the user's settings using `elgg_get_all_plugin_user_settings()`, which returns the settings in an associative array. We hard-code the toolbar choices and limit them to blog, files, settings, and the messages inbox. The labels for the checkboxes come from the language file:

```
<?php
/**
 * User defined toolbar language strings
 */
$english = array(
  "udt:instruct" => "Select which items you would like in
    your toolbar.",

  "udt:blog" => "My blog",
  "udt:settings" => "Settings",
  "udt:inbox" => "Inbox",
  "udt:files" => "My files",
);

add_translation("en", $english);
```

The end result of the view is as shown in the following screenshot:


User Defined Toolbar

Select which items you would like in your toolbar.

☐ My blog
☑ Settings
☑ Inbox
☑ My files

Just as with widgets settings, Elgg handles saving the options in the database as long as we use names with "`params[]`".

Step 3: Create the toolbar view

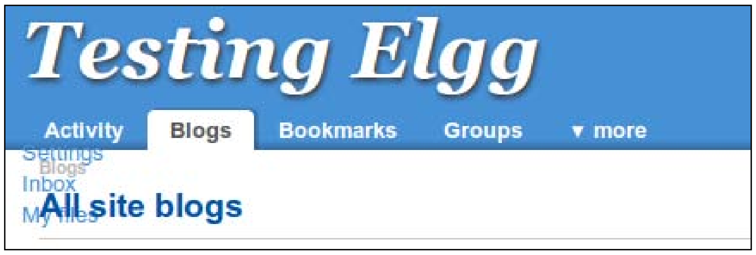
For the toolbar, we want to create a list of items that the user has selected. We retrieve the user's settings as we did in the settings view using `elgg_get_all_plugin_user_settings()`. We then loop through the settings to determine which items have been activated. The `toolbar.php` file should look like the following code:

```
<?php
/**
 * User menu
 */

$user = elgg_get_logged_in_user_entity();

$items = array(
  'blog' => "blog/owner/$user->username",
  'settings' => "settings",
  'inbox' => "messages/inbox/$user->username",
  'files' => "files/owner/$user->username"
);

$settings = elgg_get_all_plugin_user_settings($user->guid,
  'user_defined_toolbar');

if ($settings) {

  echo '<ul id="udt-toolbar">';

  foreach ($items as $name => $url) {

    if ($settings[$name]) {
      $link = elgg_view('output/url', array(
        'text' => elgg_echo("udt:$name"),
        'href' => $url,
      ));
      echo "<li>$link</li>";
    }
  }

  echo '</ul>';
}
```

The output is an unordered list of links. We personalize the links to point to that user's blogs, inbox, and files by asking for the logged in user from Elgg. The link addresses are obtained by checking the URLs for the **Mine** tab on the blog and files pages.

Step 4: Add CSS

If you viewed the toolbar on your site right now, then you would see that it is not usable because the text overlaps other page elements, as follows:

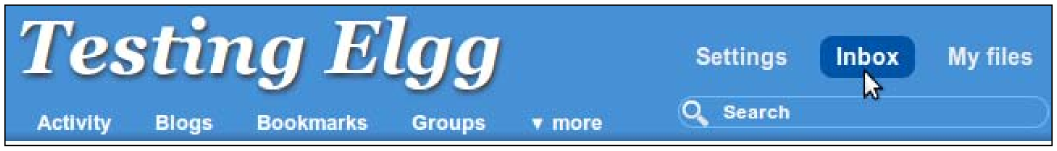


We need to add the CSS to position and style the toolbar. Open the `css` view and insert the following code:

```
.elgg-page-header h1
{
  float: left;
}
.udt-toolbar
{
  position: absolute;
  top: 25px;
  right: 0;
}
.udt-toolbar li
{
  display: inline-block;
  margin-left: 10px;
}
.udt-toolbar li a
{
  display: block;
  font-size: 1.1em;
  font-weight: bold;
  color: #eeeeee;
  padding: 4px 10px;
  border-radius: 8px;
  -webkit-border-radius: 8px;
  -moz-border-radius: 8px;
  text-decoration: none;
}
```

```
.udt-toolbar li a:hover
{
  color: #ffffff;
  background-color: #0054A7;
  text-decoration: none;
}
```

The first statement floats the site name text so that we can position our toolbar to its right. After that, we position the list, make it horizontal, and style the links to look like buttons when hovered over. The final product is as shown in the following screenshot:



Exercise

A good learning exercise related to user settings is writing a plugin that allows users to set the background color of their profile page. Because the primary CSS file is cached, it is not possible to personalize that file. Instead, you could include the CSS inline in the page like the following:

```
<style type="text/css">
  body {
    background-color: black;
    color: white;
  }
</style>
```

Instead of hard coding the CSS colors, use colors from the user settings. To reflect the owner of a profile page rather than the person viewing the page, the owner's GUID needs to be passed:

```
$guid = elgg_get_page_owner_guid();
$settings = elgg_get_all_plugin_user_settings
  ('profile_background', $guid);
```

You also need to make sure that the inline CSS is only included on profile pages. This can be done with a test of the context, as follows:

```
if (elgg_in_context('profile')) {
  // do stuff here
}
```

Lesson 7: Adding JavaScript

This lesson demonstrates using a jQuery plugin with Elgg.

Problem

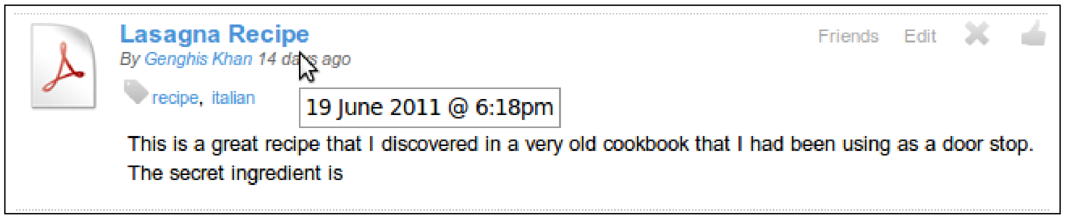
You have been looking at pages that use really slick jQuery plugins. (Do a search on "`top jquery plugins`" if you are not familiar with all the possibilities offered by jQuery plugins.) Lightboxes, tool tips, charts, animations, fancy forms – the list goes on and on. You want to add a few of these jQuery plugins to your Elgg site, but you are not sure how to accomplish this.

Solution

Many jQuery plugins only require inclusion of a JavaScript file, often a CSS file, and then a little bit of JavaScript to initialize the plugin. Elgg has functions for including external JavaScript and CSS files that make this easy.

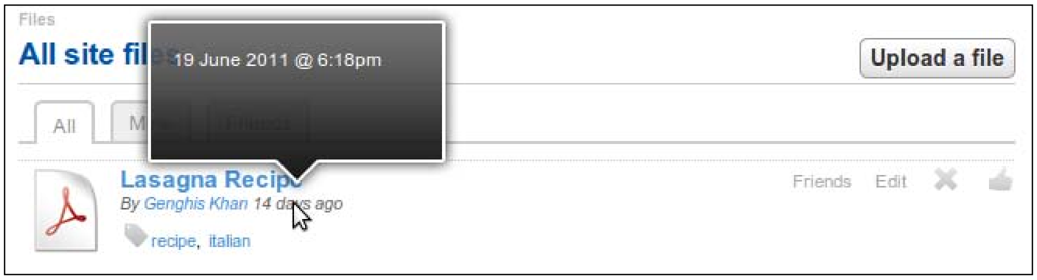
Example

In this example, we are adding a fancy tooltip to all the timestamps of content on the site. Normally, when you mouse over a timestamp, you see a tooltip created by the browser, like in the following screenshot:



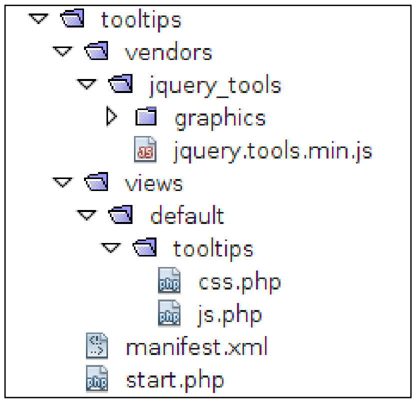
This is a great recipe that I discovered in a very old cookbook that I had been using as a door stop. The secret ingredient is

We are using the jQuery Tools library (`http://flowplayer.org/tools/tooltip/index.html`) to create tool tips that look like the following screenshot:



Step 1: Create the plugin structure

Our `tooltips` plugin has a plugin manifest, a start script, a `css` view and a `js` view, along with the files that make up the tooltips library. The tooltips library can be downloaded here: `http://flowplayer.org/tools/download/index.html` and the graphics are available here: `http://flowplayer.org/tools/img/tooltip/tooltip.zip`. We put the JavaScript file and graphics in a vendor's directory to keep it separate from the code that we are writing. The plugin structure looks like the following:



```
tooltips
  vendors
    jquery_tools
      graphics
      jquery.tools.min.js
  views
    default
      tooltips
        css.php
        js.php
  manifest.xml
  start.php
```

Step 2: Load the JavaScript file

Because we want this jQuery plugin to run on every page, we register and load it in our plugin's initialization function, as follows:

```
elgg_register_event_handler('init', 'system', 'tooltips_init');

function tooltips_init() {
  $url = 'mod/tooltips/vendors/jquery_tools/jquery.tools.min.js';
  elgg_register_js('jquery.tools', $url, 'footer');
  elgg_load_js('jquery.tools');
}
```

The `elgg_register_js()` function tells Elgg that there is a JavaScript file by the name of "`jquery.tools`" at the specified URL. We also registered the JavaScript file to be included in the footer of the page rather than in the HTML `<head>` element. This is a community-accepted best practice for performance. After registering the file, we immediately tell Elgg to load it, making it available on every page. With other scripts, you may only want to load them on particular pages. In those cases, you would call `elgg_load_js()` from the page handler or from a view that requires it.

Step 3: Add CSS and JavaScript initialization

There are two remaining items before we are done. We need to initialize the tooltips JavaScript library and style the tooltips with CSS. We accomplish both by extending core views in the initialization function, as follows:

```
elgg_register_event_handler('init', 'system', 'tooltips_init');

function tooltips_init() {
  $url = 'mod/tooltips/vendors/jquery_tools/jquery.tools.min.js';
  elgg_register_js('jquery.tools', $url, 'footer');
  elgg_load_js('jquery.tools');

  elgg_extend_view('js/elgg', 'tooltips/js');
  elgg_extend_view('css/elgg', 'tooltips/css');
}
```

We use the CSS and graphics provided from the jQuery Tools website. The `css.php` file then looks like the following:

```
<?php
/**
 * Tooltips CSS
 */

$img = 'mod/tooltips/vendors/jquery_tools/
  graphics/black_arrow.png';
$img = elgg_normalize_url($img);
?>

.tooltip {
  display: none;
  background: transparent url(<?php echo $img; ?>);
  font-size: 12px;
  height: 70px;
  width: 160px;
  padding: 25px;
  color: #fff;
}
```

Instead of hard-coding the URL of the image, we calculate it from a relative URL using the `elgg_normalize_url()` function. This function creates an absolute URL using the base URL of the site. This ensures that the code works on any site the plugin is installed on.

We initialize the tooltips jQuery plugin by extending the primary Elgg JavaScript file with the following code:

```
<?php
/**
 * Extend the main elgg.js script for tooltips initialization
 */
?>

elgg.provide('elgg.tooltips');

elgg.tooltips.init = function() {
  $("acronym[title]").tooltip();
}

elgg.register_hook_handler('init', 'system', elgg.tooltips.init);
```

The first line creates a JavaScript object for our plugin. We then define an initialization function and register that function to run when Elgg's JavaScript library performs its initialization. This should look familiar as Elgg's JavaScript initialization approach is based on its PHP code. Lastly, we call the tooltip function on all the acronym tags that have a title. This configures the tooltips JavaScript to run whenever the mouse moves over one of the timestamps.

Exercise

There are many jQuery plugins out there and using them is as easy as follows:

1. Include the jQuery plugin's JavaScript, CSS, and any images that it uses.
2. Find, modify, or add HTML required for the plugin.
3. Add an initialization script.

A good starting point is adding a `lightbox` to the image gallery page. This would enable users to view large versions of the images without leaving the page. Elgg is distributed with the FancyBox jQuery library and its files can be loaded with the following lines:

```
elgg_load_js('lightbox');
elgg_load_css('lightbox');
```

Lesson 8: Changing how Elgg does X

This lesson covers using a plugin hook. It also explains how to add administrative settings.

Problem

Groups can have blogs, but only the person who writes the blog post can edit it. You prefer that group blogs work like the pages tool that allow for collaborative editing. Or maybe you would like to create customized notification messages that are sent to your users through e-mail. You also might want to include custom information when people search your site. The common theme here is that Elgg is doing one thing and you would like it to do it differently.

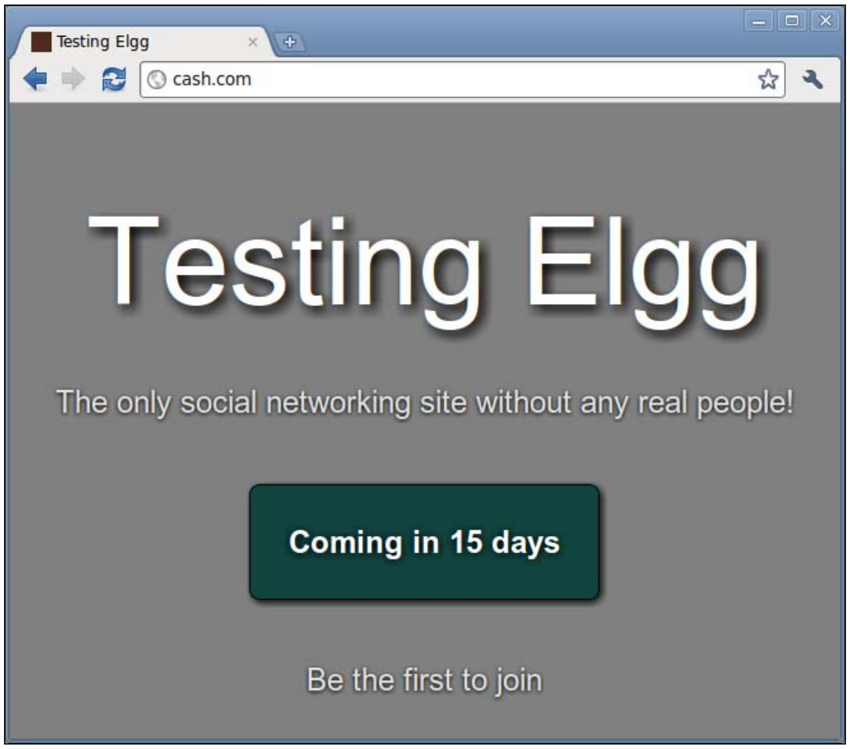
Solution

Use plugin hooks, which are very similar to events. When something occurs, a plugin hook is triggered and the functions that are registered for that plugin hook are called. The primary difference between events and plugin hooks is that plugin hook functions can return a changed result to the core. Take the `'validate, input'` plugin hook as an illustrative example. When a user submits a new blog post, the text of the blog post is sent through the `'validate, input'` plugin hook. The `htmlawed` plugin scans the text, checking if the user is trying to hack the site. If it finds suspect text, then it removes the text. We could write an additional plugin for the '`validate, input`' hook that scans for obscene language and removes it or warns the user.

Example

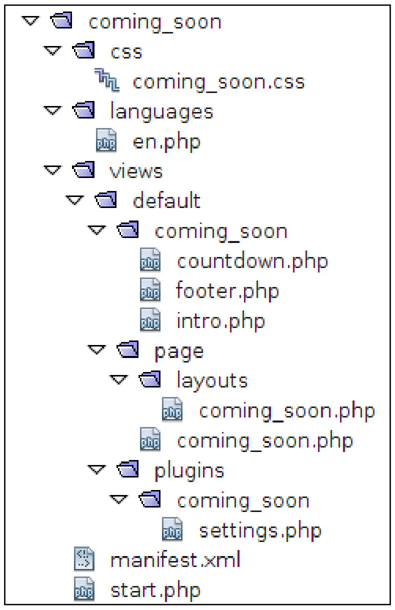
There is a list of available plugin hooks on the Elgg wiki (`http://docs.elgg.org/List_of_Plugin_Hooks`). One of the hooks listed is `'index, system'` which is triggered whenever a browser requests the front page. This hook lets plugins create a custom front page for Elgg instead of the default one provided by the engine.

We are going to use this plugin hook to create a **Coming soon** page. We have been working hard, but our site is not ready for the public. We would really like to be testing our code on our production server without anyone being able to see the site. We also want to have a page up that lets people know the site is coming soon and perhaps pique their interest. We want a page like the following screenshot:



Step 1: Create the plugin structure

The **coming_soon** plugin uses knowledge that we have gained through the previous lessons. Besides the typical plugin manifest, start script, and language file, this plugin has an administrative setting page, its own CSS file, a custom layout, and a custom page shell. Add to that three more views and we have the the following structure:



Step 2: Create the index page

After creating all the files that this plugin needs, we add code to the `start.php` file to register for the `'index, system'` plugin hook.

```
elgg_register_event_handler('init', 'system', 'coming_soon_init');

function coming_soon_init() {
  elgg_register_plugin_hook_handler('index', 'system',
    'coming_soon_index', 1);
}
```

Notice that the last parameter is `1` for the `elgg_register_plugin_hook_handler()` function. This sets the priority for the callback function ensuring that our plugin gets the first opportunity to replace the front page. The function we registered uses the custom layout and custom page shell to create the web page. It returns true to indicate that the front page has been handled. If we do not return true, then the user would see two front pages.

```
function coming_soon_index() {

  $body = elgg_view_layout('coming_soon');

  // use our own page shell
  echo elgg_view_page('', $body, 'coming_soon');

  return true;
}
```

When we pass "coming_soon" as the layout name to `elgg_view_layout()`, Elgg uses the view "`page/layouts/coming_soon`". In this view, we pull together three views to create the page: an intro that has the site title, a countdown to when the site launches, and a footer that tells the viewer about the people developing the site. The view looks like the following:

```
<?php
/**
 * Layout of coming soon front page
 */

echo '<div id="coming-wrapper">';

echo elgg_view('coming_soon/intro');
echo elgg_view('coming_soon/countdown');
echo elgg_view('coming_soon/footer');

echo '</div>';
```

We are also using a custom page shell as we do not want Elgg to include the normal header and footer on this page. The page shell contains the basic structure of the HTML web page. The following code is what ours looks like:

```
<?php
/**
 * Coming soon pageshell
 */

// Set the content type
header("Content-type: text/html; charset=UTF-8");

$title = elgg_get_site_entity()->name;
$url = 'mod/coming_soon/css/coming_soon.css';
$url = elgg_normalize_url($url);

?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
  lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html;
      charset=utf-8" />
    <title><?php echo $title; ?></title>
    <!-- include our custom css file -->
    <link rel="stylesheet" href="<?php echo $url; ?>"
      type="text/css" />
  </head>
  <body>
    <?php echo $vars['body']; ?>
  </body>
</html>
```

This is a bare bones page shell that sets the title of the page and includes our CSS file. To look at Elgg's default page shell, open the file `/views/default/page/default.php`. This is where Elgg includes most of its JavaScript and CSS. It also sets the structure for the page with sections for the topbar, header, layout area, and footer.

Step 3: Define the views

Now that the structure for our index page is in place, we can fill in the content of the three views that we are including in our layout. The three views are very simple. It is not absolutely necessary to divide them in this manner, but it is a good habit to break your HTML code into structural chunks. The intro view has the title and uses the site's description as set on the **Basic Settings** page:

```
<?php
/**
 * Introduction view
 */

$site = elgg_get_site_entity();

$title = $site->name;
$tagline = $site->description;

echo "<h1>$title</h1>";
echo "<h3>$tagline</h3>";
```

The countdown view will automatically update by the time we are done with the plugin, but for right now, we hard-code its value.

```
<?php
/**
 * Countdown view
 */

$countdown = elgg_echo('coming_soon:coming', array(27));
```

```
?>

<div id="coming-countdown">
  <span>
    <?php echo $countdown; ?>
  </span>
</div>
```

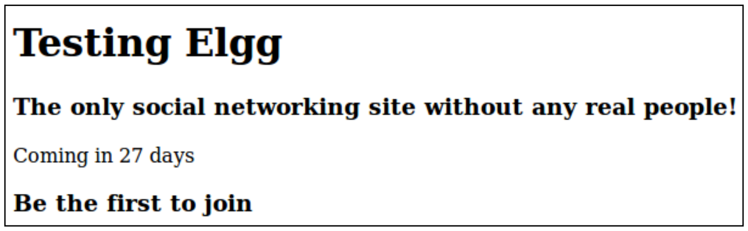
The footer is similarly simple:

```
<?php
/**
 * Footer view
 */
?>

<h3>
  <?php echo elgg_echo('coming_soon:footer'); ?>
</h3>
```

We used two language strings so we add those to `en.php`:

```
'coming_soon:coming' => 'Coming in %s days',
'coming_soon:footer' => 'Be the first to join',
```

If we activate the plugin now without the needed CSS, the front page looks like the following screenshot:



Testing Elgg

The only social networking site without any real people!

Coming in 27 days

Be the first to join

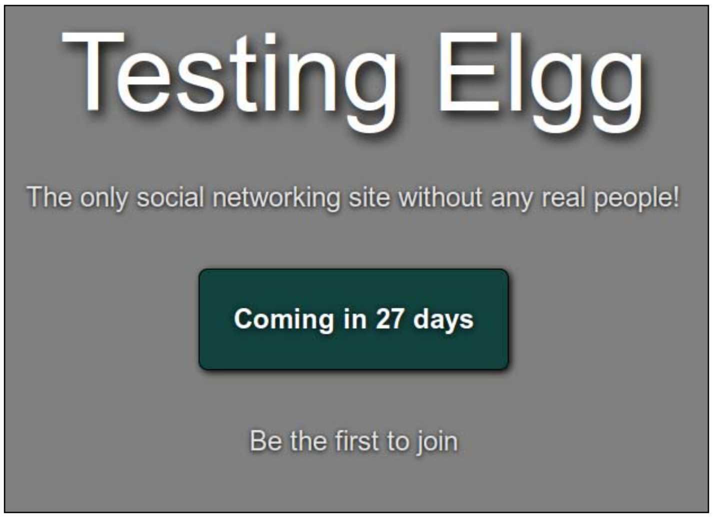
If you activated the plugin and are logged out of your site, then you can log back in by going to `http://example.org/login` (except replace `example.org` with your site address).

Step 4: Add the CSS

We add our CSS to the `css/coming_soon.css` file that we included in the page shell, as follows:

```
body {
  background-color: #808080;
  font-family: "Helvetica Neue", Arial, Helvetica, sans-serif;
}
h1 {
  color: white;
  text-shadow: black 0.05em 0.05em 0.15em;
  font-size: 6em;
  line-height: 2em;
}
h3 {
  color: #e1e3e1;
  text-shadow: 0px 1px 5px black;
  font-size: 1.5em;
}
#coming-wrapper {
  width: 640px;
  margin: 30px auto;
  text-align: center;
}
#coming-countdown {
  margin: 80px 0;
}
#coming-countdown span {
  color: white;
  background-color: #12443F;
  border: 1px solid black;
  padding: 30px;
  font-size: 1.5em;
  font-weight: bolder;
  text-shadow: 0px 1px 10px black;
  border-radius: 8px;
  -webkit-border-radius: 8px;
  -moz-border-radius: 8px;
  box-shadow: 2px 2px 8px black;
  -moz-box-shadow: 2px 2px 8px black;
  -webkit-box-shadow: 2px 2px 8px black;
}
```

We use a few newer CSS elements such as text-shadow and border-radius. Versions of Internet Explorer before IE9 ignore these elements, but modern browsers display them. Our front page looks like what we want:



Step 5: Add a plugin setting

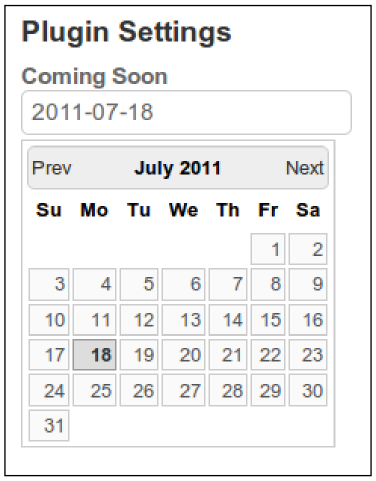
We do not want to edit the code each day to change the countdown value. Instead, we set the launch date as a plugin setting and calculate how many days are left with some PHP code. The view for our plugin setting is "`plugins/coming_soon/settings`". It works exactly like the user settings view in that Elgg automatically detects its existence and handles saving the data for us.

The plugin setting view uses a date picker for selecting the launch date:

```
 <?php
/**
 * Plugin settings
 */

// if user has not set date, show today on calendar
$release_date = time();
if (isset($vars['entity']->release_date)) {
  $release_date = $vars['entity']->release_date;
}

// add a date picker
$options = array(
  'name' => 'params[release_date]',
   'value' => $release_date,
  'timestamp' => true,
);
echo elgg_view('input/date', $options);
```

The plugin entity is passed to the plugin settings view and is available as `$vars['entity']`.

Elgg knows to initialize the data picker when the "`input/date`" view is used. If we visit the `Coming Soon` settings page, the date picker looks like the following image:



Su	Mo	Tu	We	Th	Fr	Sa
					1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31						

Once the plugin settings view is finished, we head back to the countdown view to calculate the number of days until launch:

```
<?php
/**
 * Countdown view
 */

$release_date = elgg_get_plugin_setting('release_date',
  'coming_soon');

// number of seconds until release
$diff = $release_date - time();

// round up to number of days
$day = 24 * 60 * 60;
```

```
$num_days = ceil($diff/$day);

$countdown = elgg_echo('coming_soon:coming', array($num_days));
?>

<div id="coming-countdown">
  <span>
    <?php echo $countdown; ?>
  </span>
</div>
```

We requested that Elgg store the date as a Unix timestamp (the number of seconds that has elapsed since January 1, 1970) by setting the timestamp parameter as true for the `'input/date'` view. We subtract the current time as a Unix timestamp to calculate how many seconds to the launch of our site. That is not very useful to potential users so we convert that into days. Now the passing of each day lowers the countdown for our site launch.

Exercise

This example did not modify the data passed by the plugin hook to your function. To try writing a plugin that does, we use the `'view, *'` plugin hook. This hook sends your function the output of a view. You can add, remove, or replace content from the HTML of the view and then return it at the end of your function. One example is registering for the `'view, output/longtext'` view. This view is used for displaying the body of a blog post or the content of a page. In your function, you could check for certain keywords and wrap those words with links to pages on your site. If your site was for geography students, then you could catch the names of countries and add links related to that country:

```
$country = 'Andorra';
$link = "<a href=
  'http://wikipedia.org/wiki/$country'>$county</a>";
$text = str_replace($country, $link, $text);
```

After you have made the appropriate replacements, return the new string and Elgg will use it as the output of the view.

Lesson 9: Collecting and storing data

This is the longest of the lessons. It demonstrates how to create forms, how to create an action, and how to save the data using Elgg's data model.

Problem

You know how to store settings for plugins, widgets, and users, but how do you collect and store more complicated data? You want your users to write testimonials about your site that you can rotate on the front page. Or you would like to have a custom feedback system that lets users report problems or offer suggestions. Maybe you are thinking that it would be great if your users could post their favorite poems or excerpts from books and then describe why they like them. The one common issue with all of these is how to gather and store the data.

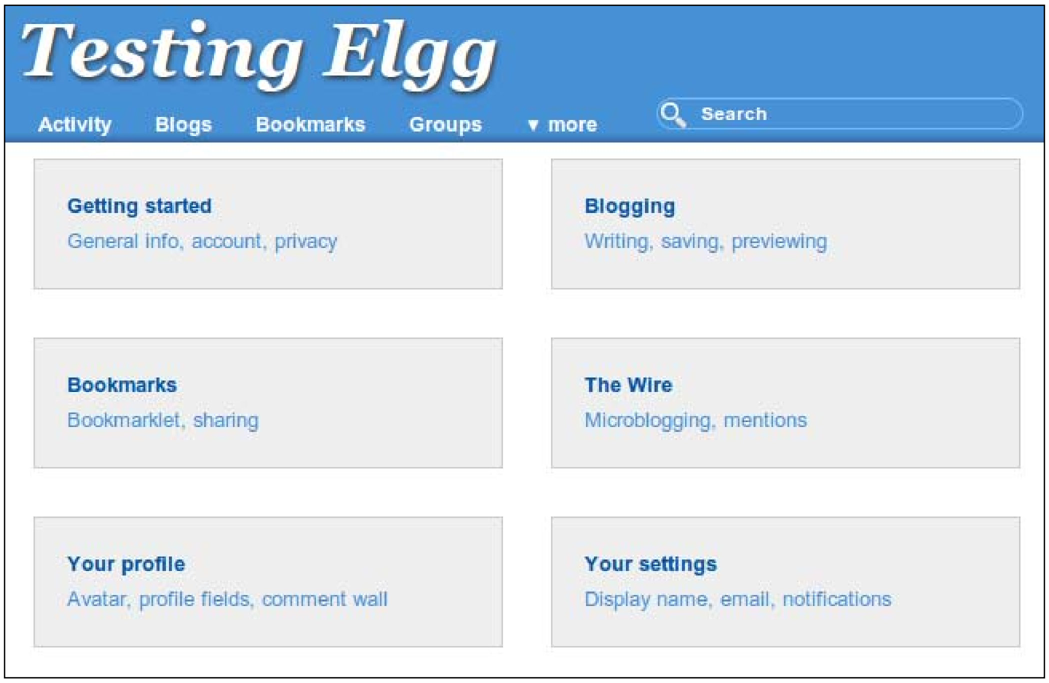
Solution

A form is used to collect data from users on websites. When a user registers on an Elgg site, the user's name, e-mail address, and password are all entered into a form. The form includes an address for the browser to send the data when the user clicks on the submit button. This address is called the **action**. Elgg has an action system to manage this process. Elgg also sends the user to a new page after submitting a form. Let's take a look at creating forms and actions in this last lesson.

Example

In this example, we are creating a help system for our site. We want a categorized page of help topics that makes it easy for users to find the information they are looking for. Our help system will have an administrative interface for adding new help content. This is where the forms and actions come in. Our administrator fills in the information about new help topics and submits it. We want Elgg to save the content and dynamically update the help pages.

When we are all done, the main help page will look like the following screenshot:



Getting started
General info, account, privacy

Blogging
Writing, saving, previewing

Bookmarks
Bookmarklet, sharing

The Wire
Microblogging, mentions

Your profile
Avatar, profile fields, comment wall

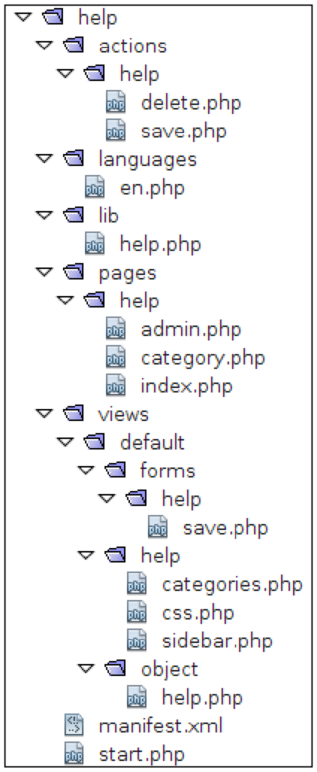
Your settings
Display name, email, notifications

Overview

This is the most extensive of the tutorials in this chapter. It introduces creating forms and the actions to process a form's data. This plugin is the first to use Elgg's data model. For a high-level overview of the data model, read *Appendix A*. The rest of the plugin builds on what you learned in the previous chapter and this one.

Step 1: Create the plugin structure

The `help` plugin has more components than any other plugin that we have written. Besides the plugin files we have seen in the past, this one also has a library file to store common functions, action files, and three page files are put in a separate directory to keep everything organized. When we are finished we will have 14 files structured as follows:



```
help
  actions
    help
      delete.php
      save.php
  languages
    en.php
  lib
    help.php
  pages
    help
      admin.php
      category.php
      index.php
  views
    default
      forms
        help
          save.php
      help
        categories.php
        css.php
        sidebar.php
      object
        help.php
  manifest.xml
  start.php
```

Step 2: Create the main help page

We are using a page handler for our pages with addresses like `http://example.org/help`. We register the page handler in our plugin's initialization function, as follows:

```
elgg_register_event_handler('init', 'system', 'help_init');

function help_init() {

  // page handler for the help system
  elgg_register_page_handler('help', 'help_page_handler');
}
```

We also create a skeleton page handler function, as follows:

```
function help_page_handler($page) {

  $pages_dir = elgg_get_plugins_path() . 'help/pages/help';
```

```
    // send everything to main index
    require "$pages_dir/index.php";

    return true;
}
```

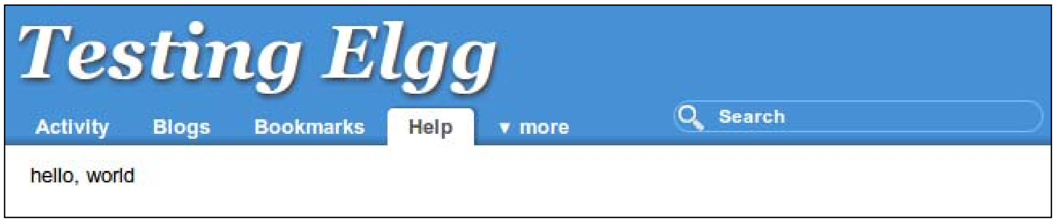
This loads our main index page when a user visits any page starting with `/help/`. We keep our main index page very simple for now while we set everything else up.

```
<?php
/**
 * Main help index page - list help categories
 */

$title = elgg_echo('help:categories');

$content = "hello, world";

$body = elgg_view_layout('one_column', array('content' =>
  $content));

echo elgg_view_page($title, $body);
```

To give people access to this page, we add a link to the site menu in the plugin's initialization function:

```
    // add menu item for help
    $item = new ElggMenuItem('help', elgg_echo('help'), 'help');
    elgg_register_menu_item('site', $item);
```

If we promote the **Help** menu item using the administrative **Menu Items** page of Elgg, then our main help page looks like the following screenshot:



hello, world

Step 3: Create the categories

The help categories rarely change so we will hard-code them in our library file, `help.php`. We are using the library file to demonstrate a possible method for organizing code that becomes necessary as plugins become more complicated. We put the categories in a function so that we can easily access them anywhere. Each category has a code and a title. The titles are stored in the language file in case we need to support more than one language. Put this function in `lib/help.php`, as follows:

```
function help_get_categories() {
  $codes = array(
    'getting_started',
    'blogging',
    'bookmarks',
    'thewire',
    'profile',
    'settings',
  );
  $categories = array();
  foreach ($codes as $code) {
    $categories[$code] = elgg_echo("help:title:$code");
  }

  return $categories;
}
```

The corresponding language strings are as follows:

```
'help:title:getting_started' => 'Getting started',
'help:title:blogging' => 'Blogging',
'help:title:bookmarks' => 'Bookmarks',
'help:title:thewire' => 'The Wire',
'help:title:profile' => 'Your profile',
'help:title:settings' => 'Your settings',
```

We register the library with Elgg in the plugin initialization function and tell Elgg to load it on every page:

```
$lib = elgg_get_plugins_path() . 'help/lib/help.php';
elgg_register_library('help', $lib);
elgg_load_library('help');
```

If we had a very large library, then we could selectively load the library depending on whether it was needed for better performance.

Step 4: Add an administration page

The first step to adding the administrative page is adding to the page handler. We want the admin page to be `http://example.org/help/admin/` so 'admin' will stored in $page[0]. We add a `switch` statement to direct the requests and an `if` statement to catch the requests to `http://example.org/help`:

```
function help_page_handler($page) {

  $pages_dir = elgg_get_plugins_path() . 'help/pages/help';

  // this is a url like http://example.org/help
  if (count($page) == 0) {
    $page[0] = 'index';
  }

  switch ($page[0]) {
    // help/admin
    case 'admin':
      require "$pages_dir/admin.php";
      break;
    // index page or unknown requests
    case 'index':
    default:
      require "$pages_dir/index.php";
      break;
  }

  return true;
}
```

The code for the administrative page is as shown in the following code snippet:

```
<?php
/**
 * Create help topics page
 */

// only admins should see this page
admin_gatekeeper();

// context will be help so we need to set to admin
elgg_set_context('admin');

$title = elgg_echo('help:admin');

$content = elgg_view_title($title);

$content .= elgg_view_form('help/save');

// use special admin layout
$body = elgg_view_layout('admin', array('content' => $content));

echo elgg_view_page($title, $body, 'admin');
```

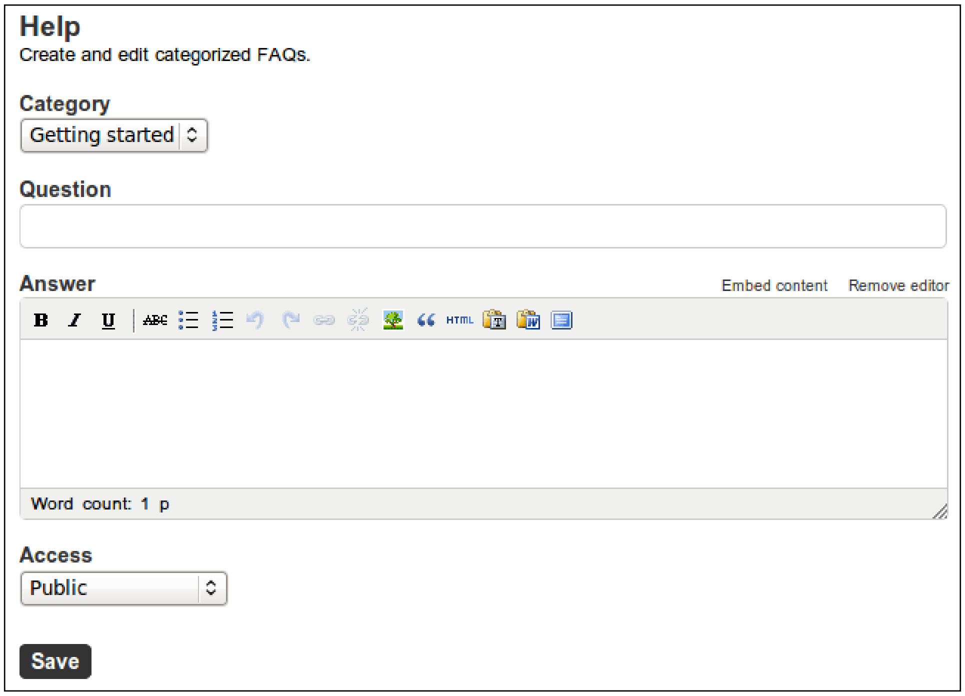
The first line is to ensure that only site administrators can view the page. Anyone else trying to look at the page ends up being sent to the front page of the site. As this is an administrative page, we also set the context to be `'admin'` and use the administrative layout and page shell.

The content of the page is its title and the form for saving help topics. The function `elgg_view_form()` knows to get the body of the form from the view "`forms/help/save`". It also configures the form to submit its data to the "`help/save`" action which we create in step 6.

Step 5: Create the form body

The form bodies that we have written so far have been very simple: the widget edit form, the plugin settings form, and the plugin user settings form. This form has four input fields and a submit button. The views for all of these controls start with "input" and you can see what input views are available by looking in the core directory `views/default/input/`.

Our form has input fields for the category, question, answer, and access level and will look like the following screenshot when we are done:



Help

Create and edit categorized FAQs.

Category

Getting started

Question

Answer

Word count: 1 p

Access

Public

Add the following code to the "`forms/help/save`" view to create this form:

```
<?php
/**
 * help/save form body
 *
 */

$instructions = elgg_echo('help:admin:instruct');

$categories = help_get_categories();
$category_input = elgg_view('input/dropdown', array(
  'name' => 'category',
  'options_values' => $categories,
));

$question_input = elgg_view('input/text', array(
  'name' => 'question',
));

$answer_input = elgg_view('input/longtext', array(
  'name' => 'answer',
));

$access_input = elgg_view('input/access', array(
  'name' => 'access_id',
));

$button = elgg_view('input/submit', array(
  'value' => elgg_echo('save')
));

echo <<<HTML

<div>$instructions</div>
<div>
  <label>Category</label><br />
  $category_input
</div>

<div>
  <label>Question</label><br />
  $question_input
```

```
</div>

<div>
  <label>Answer</label>
  $answer_input
</div>

<div>
  <label>Access</label><br />
  $access_input
</div>

<div class="elgg-foot">
  $button
</div>
HTML;
?>
```

The possible categories are retrieved using the `help_get_categories()` function that we previously created. We use a drop-down control for them. The questions and answers get textboxes with the answers getting the longer one because we expect that they will have more text. The access control determines who can see the help topic. If we wanted to edit one several times before making it public, then we could set the access level to `Private`. We did not use language string for the labels, to make the code easier to follow.

Clicking on the save button will result in an error message saying `"The requested action 'help/save' is not defined in the system."` This is because we have not registered the action with Elgg yet.

Step 6: Create the save action

We register the save action in the plugin's initialization function, as follows:

```
$base = elgg_get_plugins_path() . 'help/actions/help';
elgg_register_action("help/save", "$base/save.php", "admin");
```

This tells Elgg where to find the action and that only administrators can use it.

In the action file, we get the values submitted in the form from Elgg using `get_input()`. This function automatically filters the data through the `htmlawed` plugin looking for possible hacking attacks. After we save the help topic to the database, we register a status message for later display and forward the user back to the help administration page.

```
<?php
/**
 * Save a help topic
 */

// get the form values
$category = get_input('category');
$question = get_input('question');
$answer = get_input('answer');
$access_id = get_input('access_id');

// save the question and queue status message
$result = help_save_topic($question, $answer, $category, $access_id);
if (!$result) {
  register_error(elgg_echo('help:error:no_save'));
} else {
  system_message(elgg_echo('help:status:save'));
}

forward(REFERER);
```

The function that saves the topic has not been created yet, but we want to put that function in the help library. Open the library file and insert the following function:

```
function help_save_topic($question, $answer, $category,
  $access_id) {
  $help = new ElggObject();
  $help->subtype = 'help';
  $help->title = $question;
  $help->description = $answer;
  $help->access_id = $access_id;

  $help->category = $category;


  $guid = $help->save();
  if (!$guid) {
    return false;
  }

  return true;
}
```

This function saves the help topic as an ElggObject. This class supports storing `subtype`, `title`, `description`, and `access_id`. It also supports storing arbitrary data which are called **metadata**. The key difference for plugin authors is that the save method must be called for storing native attributes (such as title). The save method does not need to be called for metadata as it is automatically saved as it is created and updated. In this example, we could have moved the line that sets the category after saving the help object. If you want to read more about how ElggObjects and the Elgg data model work, then see *Appendix A* and the Elgg wiki (`http://docs.elgg.org/`).

If we submit the form now by clicking on the save button, then Elgg saves the information in the database, but we have no way of viewing it yet.

Step 7: Create the help category page

In this step, we are creating a page that lists the topics for a particular category. The address for one of those pages is `http://example.org/help/category/<category code>`. To support this, we must add more code to the `switch` statement in the page handler function, as follows:

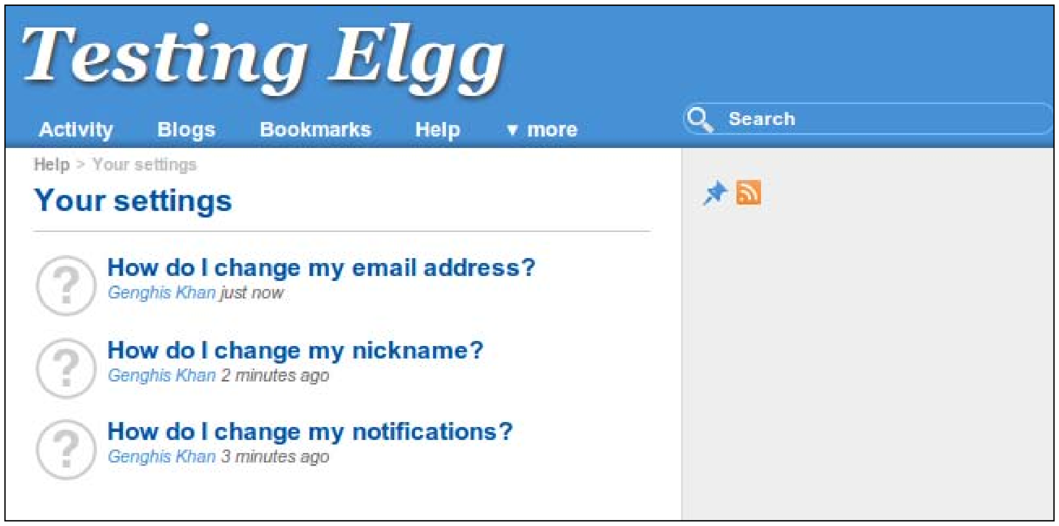
```
function help_page_handler($page) {

  $pages_dir = elgg_get_plugins_path() . 'help/pages/help';

  if (count($page) == 0) {
    $page[0] = 'index';
  }

  switch ($page[0]) {
    // help/admin
    case 'admin':
      require "$pages_dir/admin.php";
      break;
    // help/category/<category>
    case 'category':
      set_input('category', $page[1]);
      require "$pages_dir/category.php";
      break;
    // index page or unknown requests
    case 'index':
    default:
      require "$pages_dir/index.php";
      break;
  }

  return true;
}
```

Notice how we use `set_input()` to set the parameter "`category`". We do this so the script `category.php` can access the category being requested. In `topic.php`, we write the code to grab the category and then list the topics for that category, as shown in the following code snippet:

```
<?php
/**
 * List all topics (questions) in a category
 */

// category is passed from page handler
$category = get_input('category', 'getting_started');

$title = elgg_echo("help:title:$category");

elgg_push_breadcrumb(elgg_echo('help'), 'help');
elgg_push_breadcrumb($title);

// get the topics for this category
$options = array(
  'type' => 'object',
  'subtype' => 'help',
  'metadata_name' => 'category',
  'metadata_value' => $category,
  'limit' => 20,
  'full_view' => true,
  'list_class' => 'help-list',
);
$content = elgg_list_entities_from_metadata($options);

$params = array(
  'content' => $content,
  'title' => $title,
  'filter' => false,
);
$body = elgg_view_layout('content', $params);

echo elgg_view_page($title, $body);
```

Because we saved the category as metadata on our ElggObject, we retrieve the help objects by requesting all help objects with metadata named 'category' and having the value of `$category`. There are several other `elgg_get_entities*` functions that make writing powerful plugins easy. Describing them is out of the scope of this book, but you can find more information in the API reference or the Elgg wiki.

If you have not saved a few help topics, then now is a great time to do that. When you are done, type the address in your web browser to view one of the categories. If you created topics for the settings category, then the address would be `http://example.org/help/category/settings`. A category page should look like the following screenshot:



Your settings

How do I change my email address?
Genghis Khan just now

How do I change my nickname?
Genghis Khan 2 minutes ago

How do I change my notifications?
Genghis Khan 3 minutes ago

Because we did not define a view for displaying a help topic, the `elgg_list_entities_from_metadata()` function used a default view that only shows the title and the time of creation.

Step 8: Create the help object view

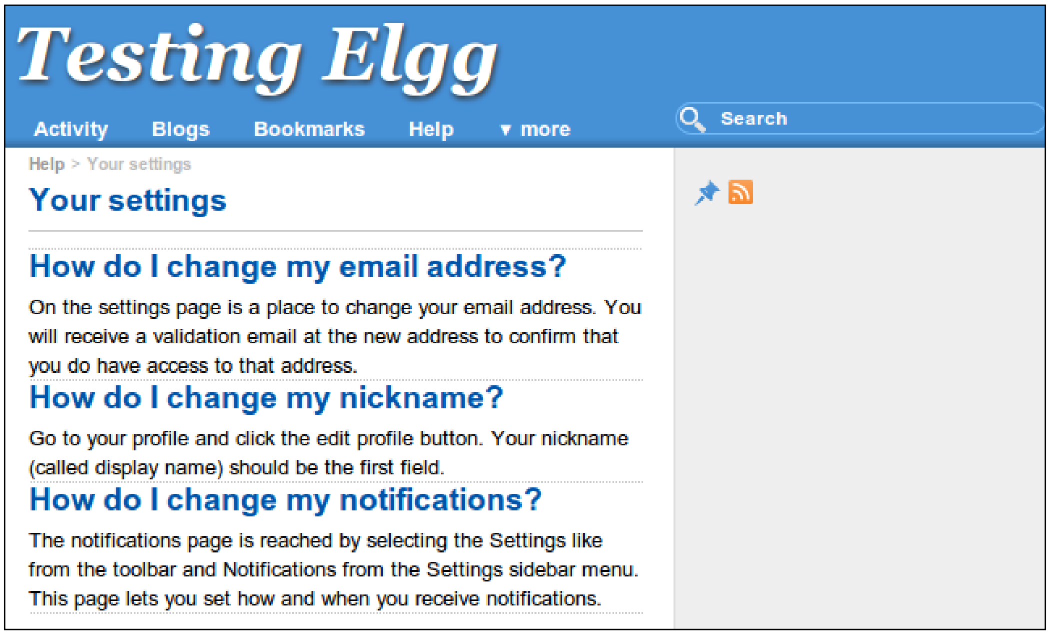
Because we saved our help topic as an ElggObject with a subtype of "`help`", Elgg looks for a `'object/help'` view. Let's define that so we can have a useful category listing page, as follows:

```
<?php
/**
 * Entity view for a help topic
 * Type: object Subtype: help
 */

$item = $vars['entity'];
$question = $item->title;
$answer = $item->description;

// full view
```

```
if ($vars['full_view']) {

  $body = elgg_view('output/longtext', array(
    'value' => $answer,
    'class' => 'mtn',
  ));

  echo <<<HTML
<div class="help-item" id="$item->guid">
  <h2>$question</h2>
  $body
</div>
HTML;

}
```

If you check the parameters that we passed to `elgg_list_entities_from_metadata()`, then you will see that we asked for the full view of the help topic. In the preceding view code, we check if it is the full view being requested and then display the object. The full view typically displays all the information about an object. When you view a blog with its comments, that is a full view. When full view is not requested, object views tend to display the title and an excerpt of the content.

Our page is much better and only needs some minor styling to be finished.



Your settings

How do I change my email address?

On the settings page is a place to change your email address. You will receive a validation email at the new address to confirm that you do have access to that address.

How do I change my nickname?

Go to your profile and click the edit profile button. Your nickname (called display name) should be the first field.

How do I change my notifications?

The notifications page is reached by selecting the Settings like from the toolbar and Notifications from the Settings sidebar menu. This page lets you set how and when you receive notifications.

Step 9: Add the help sidebar

We want to provide a quick listing of all the topics for a category in the sidebar. We define a sidebar view and call that view in the category page. Modify `category.php` to get the sidebar and pass it on to the content layout:

```
// create the sidebar
$vars = array('category' => $category);
$sidebar = elgg_view('help/sidebar', $vars);

$params = array(
  'content' => $content,
  'sidebar' => $sidebar,
  'title' => $title,
  'filter' => false,
);
$body = elgg_view_layout('content', $params);
```

The "`help/sidebar`" view uses an aside module to list the various topics for that category:

```
<?php
/**
 * Sidebar for a category's topics
 *
 * @uses $vars['category']
 */

$category = $vars['category'];

$heading = elgg_echo('help:topics');

$options = array(
  'type' => 'object',
  'subtype' => 'help',
  'metadata_name' => 'category',
  'metadata_value' => $category,
  'limit' => 0,
  'full_view' => false,
  'list_class' => 'help-list',
);
$body = elgg_list_entities_from_metadata($options);

echo elgg_view_module('aside', $heading, $body);
```

It also uses `elgg_list_entities_from_metadata()`, but turns off full view. We need to go back to the "`object/help`" and add an `else` to the `if` statement:

```
if ($vars['full_view']) {
  // snipped out full view code
} else {
  // summary view is just a link
  $url = "help/topic/$item->category#$item->guid";
  echo elgg_view('output/url', array(
    'href' => $url,
    'text' => $question,
  ));
}
```

Step 10: Add CSS for the topic listing page

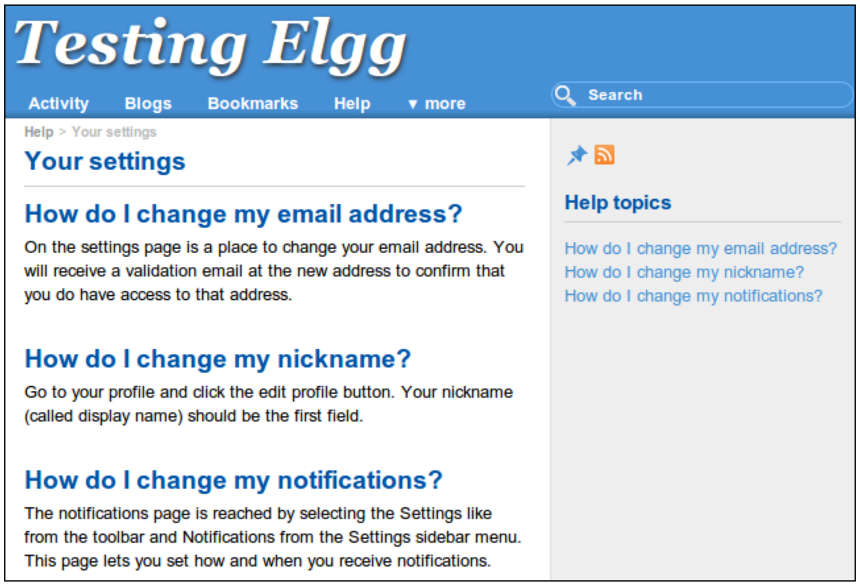
We extend the main CSS view in our plugin's initialization function, as follows:

```
  elgg_extend_view('css/elgg', 'help/css');
```

We also add some CSS to remove the lines between topics:

```
.help-list, .help-list > li {
  border: none;
}
```

Our category page looks much better, as shown in the following screenshot:



Your settings

How do I change my email address?

On the settings page is a place to change your email address. You will receive a validation email at the new address to confirm that you do have access to that address.

How do I change my nickname?

Go to your profile and click the edit profile button. Your nickname (called display name) should be the first field.

How do I change my notifications?

The notifications page is reached by selecting the Settings like from the toolbar and Notifications from the Settings sidebar menu. This page lets you set how and when you receive notifications.

Help topics

How do I change my email address?
How do I change my nickname?
How do I change my notifications?

Step 11: Finish the main help page

The last step in this example is adding a list of the different categories to the main help page. First, we replace the content variable in the main page:

```
<?php
/**
 * List the help categories
 */

$title = elgg_echo('help:categories');

$content = elgg_view('help/categories');

$body = elgg_view_layout('one_column', array('content' =>
  $content));

echo elgg_view_page($title, $body);
```

Next, we define the "`help/categories`" view:

```
<?php
/**
 * List of all help categories
 */

$categories = help_get_categories();

// create unordered list that we mark-up with CSS
echo "<ul id='help-categories'>";
foreach ($categories as $code => $title) {
  $blurb = elgg_echo("help:blurb:$code");

  $url = "/help/category/$code";
  $url = elgg_normalize_url($url);

  $text = "<h4>$title</h4><span>$blurb</span>";

  echo "<li><a href=\"url\">$test</a></li>";
}
echo "</ul>";
```

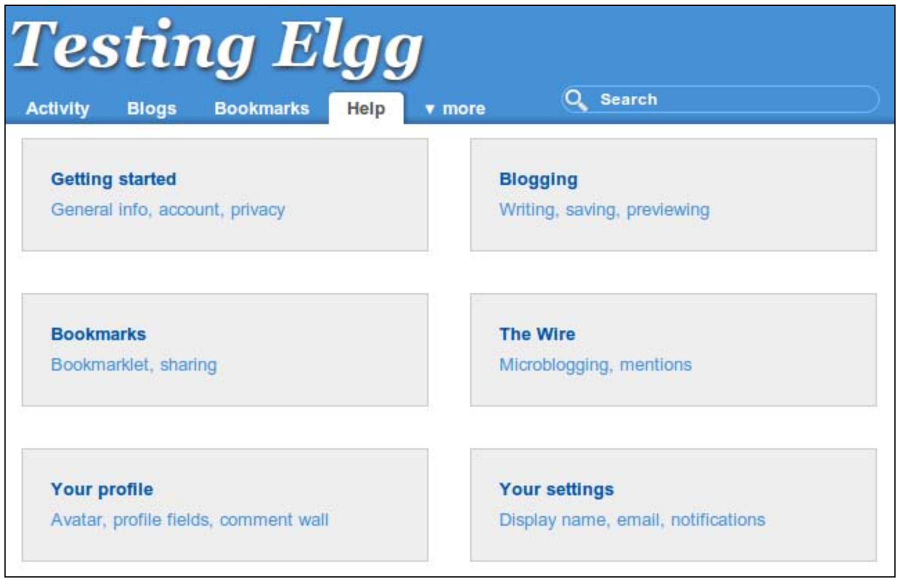
The blurb text appears underneath each category title and explains what is in the category. The blurbs are set in the language file:

```
// category blurbs
'help:blurb:getting_started' => 'General info, account,
privacy',
'help:blurb:blogging' => 'Writing, saving, previewing',
'help:blurb:bookmarks' => 'Bookmarklet, sharing',
'help:blurb:thewire' => 'Microblogging, mentions',
'help:blurb:profile' => 'Avatar, profile fields, comment wall',
'help:blurb:settings' => 'Display name, email, notifications',
```

Finally, we add some CSS for this page:

```
.help-categories li
{
  float: left;
  margin: 0 30px 30px 0;
}
.help-category
{
  float: left;
  display: block;
  width: 252px;
  padding: 20px;
  border: 1px solid #cccccc;
  background-color: #eeeeee;
}
.help-category:hover
{
  text-decoration: none;
  background: none;
}
.help-category span
{
  display: block;
  margin-top: 4px;
  color: #4690D6;
}
```

This gives us the help index page that we wanted, as shown in the following screenshot:



Getting started
General info, account, privacy

Blogging
Writing, saving, previewing

Bookmarks
Bookmarklet, sharing

The Wire
Microblogging, mentions

Your profile
Avatar, profile fields, comment wall

Your settings
Display name, email, notifications

Exercise

How about editing or deleting help topics? We have left that as an exercise for the reader. But if you get stuck, it is included with the downloadable code. You could also extend this plugin by combining it with the sidebar tips and the welcome widget plugins.

If you want to continue to work with forms and actions, then you could use this example as a pattern to create any type of plugin where the user gets to enter data, and the data is stored and displayed. Try creating a testimonial plugin that allows users to praise your site. After you have that written, integrate it with a custom front page plugin or use one from the Elgg community site to display a random testimonial on your front page.

Summary

This chapter guided you through the development of nine different plugins. Each plugin taught you one or two new concepts that you can use when customizing your Elgg site. Whenever you need to add a plugin setting or create a form, you can return to this chapter and use the examples as a template. As you write more plugins, solving these types of problems will become easier for you. You will not need to look up the names and parameters of functions or search for the right input view. Instead, you can focus on writing really cool plugins.

If your goal is to write more elaborate plugins, then you will need to learn the Elgg data model (entities, metadata, annotations, and relationships). We briefly touched on entities (ElggObject is one) and metadata, but you will find a fuller explanation of the data model in *Appendix A*. Other topics not covered in this chapter include adding to the activity stream, sending notifications, using Ajax, and using the web services API. Besides *Appendix A*, there is further information on the Elgg wiki and, of course, the source code itself is a valuable reference.

9 Theming Elgg

In Elgg, a theme sets the visual design of your website: its layout, color scheme, graphics, typography, and behavior. The first thing you notice when visiting a website is its visual design. If it uses flashing text and rainbow colors, then you assume the site is stuck in the 1990s. In fact, you may assume the site's content is just as stale as its design and not give the site a second look. This is one reason why using the right theme for your site is so important. People quickly judge websites based on appearance.

A theme does more than make a pretty picture. It also creates the user interface of your website. It draws the user's attention to the important information. It makes it easy to navigate the site and perform common tasks. When people look at your site, you want them thinking *slick* or *clean*. When they use your site, *intuitive*, *efficient*, or *fun* should be the top descriptions.

This chapter teaches the basics of creating an Elgg theme. It is written for those who have experience in HTML and CSS, but are new to Elgg. The chapter covers the following topics:

- Views system
- CSS framework
- JavaScript framework
- Comparison between theming for WordPress and Elgg
- Example theme
- Mobile theme

What you need to know

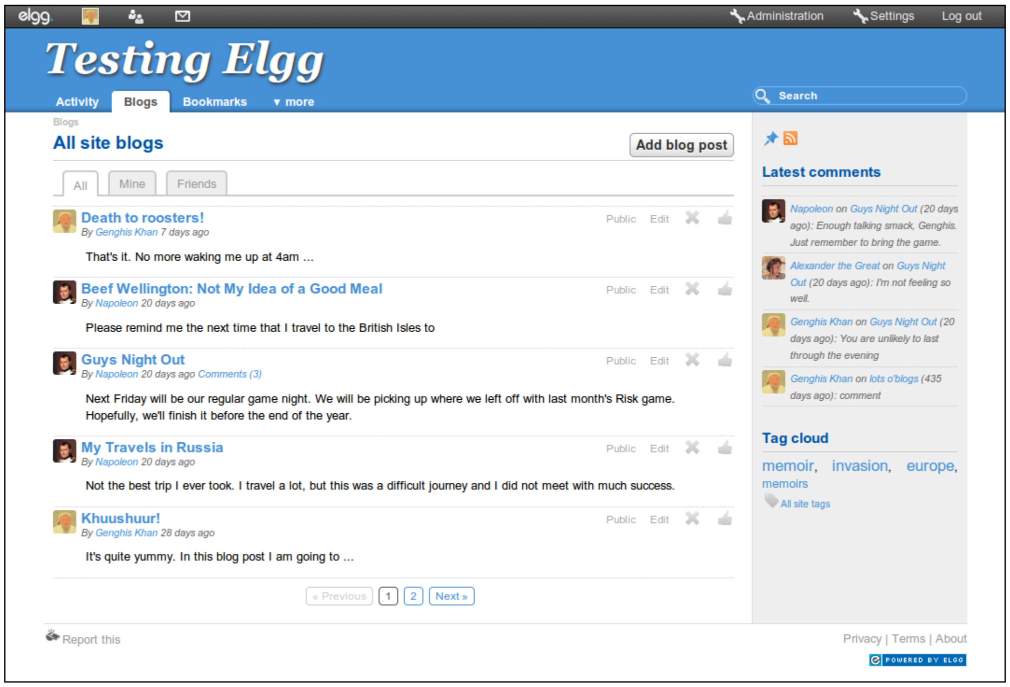
Theming absolutely requires knowledge of HTML and CSS. If you do not have experience writing HTML or CSS, then you should work through a book on HTML and CSS. You can create an entire theme using just CSS. Creating more advanced themes involves writing PHP and JavaScript (especially jQuery). Because a theme is a plugin in Elgg, at a minimum, you will need to know how to set up a plugin's structure, override views, and extend views. These concepts are reviewed in this chapter, but *Chapters 7* and *8* provide more detailed introductions to these topics.

Theming basics

This section contains reference material useful for understanding Elgg's theming system.

Elgg's default theme

In the previous chapters, you have seen many screenshots of Elgg's default theme. It is not a plugin that can be removed, but is built into Elgg. Theming is the process of replacing elements of the default theme. In this section, we provide an overview of the four main components of the theme: HTML, CSS, JavaScript, and graphics.



Death to roosters!
By Genghis Khan 7 days ago
That's it. No more waking me up at 4am ...

Beef Wellington: Not My Idea of a Good Meal
By Napoleon 20 days ago
Please remind me the next time that I travel to the British Isles to

Guys Night Out
By Napoleon 20 days ago Comments (3)
Next Friday will be our regular game night. We will be picking up where we left off with last month's Risk game. Hopefully, we'll finish it before the end of the year.

My Travels in Russia
By Napoleon 20 days ago
Not the best trip I ever took. I travel a lot, but this was a difficult journey and I did not meet with much success.

Khuushuur!
By Genghis Khan 28 days ago
It's quite yummy. In this blog post I am going to ...

Latest comments

Napoleon on Guys Night Out (20 days ago): Enough talking smack, Genghis. Just remember to bring the game.

Alexander the Great on Guys Night Out (20 days ago): I'm not feeling so well.

Genghis Khan on Guys Night Out (20 days ago): You are unlikely to last through the evening

Genghis Khan on lots o'blogs (435 days ago): comment

Tag cloud

memoir, invasion, europe, memoirs

All site tags

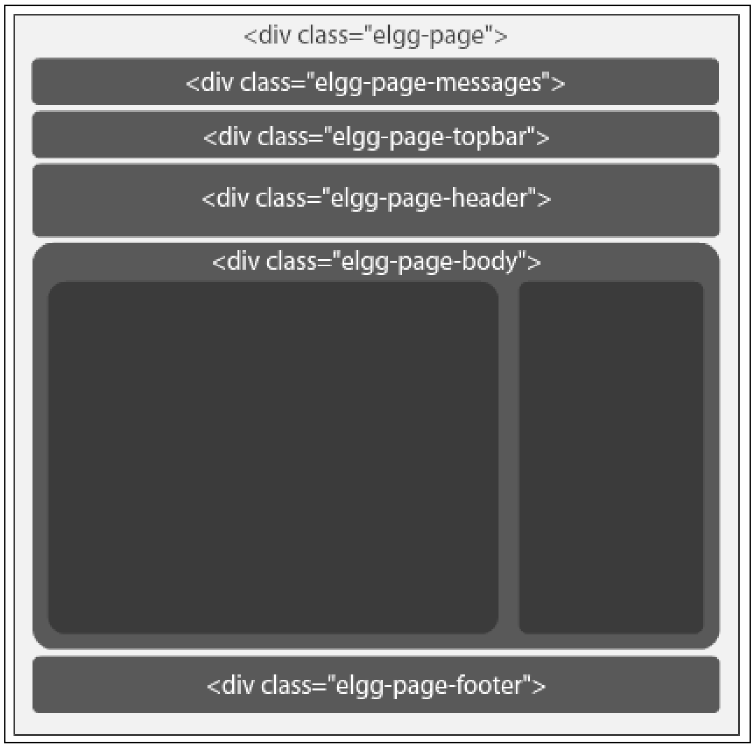
HTML

Elgg uses semantic HTML (sometimes called **Plain Old Semantic HTML** or **POSH**). With semantic HTML, the markup describes the meaning of the content rather than the presentation of it. The following are a few characteristics of semantic HTML:

- Tables are used to describe tabular data, not for layouts
- Menus tend to be lists of links so a list element is used
- Paragraph tags are used to mark paragraphs, not for spacing purposes
- Semantic CSS classes (using a class of `error` instead of `red`)

A primary advantage of this approach is that it is more intuitive to create and easier to maintain CSS when HTML describes meaning rather than presentation.

The top level of Elgg's HTML structure is called the page shell. The views that create the page shells are located in `/views/default/page/`. The primary page shell for Elgg is the view `page/default` and the page shell for the administration area is `page/admin`. The page shell assembles the main elements of a page. The default page shell includes the topbar, header, main content body, and footer. In addition, there is a `div` for status messages. It is organized as displayed in the following image:

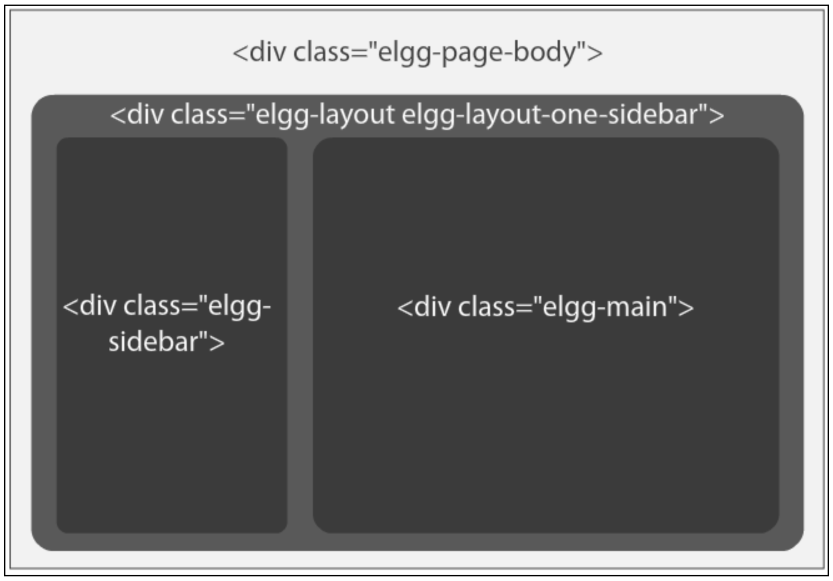


To increase the flexibility of the HTML for theming, the top-level elements consist of an outer and inner `div`. Consider the header as in the following example:

```
<div class="elgg-page-header">
  <div class="elgg-inner">
    <?php echo elgg_view('page/elements/header', $vars); ?>
  </div>
</div>
```

Each of the top-level elements has a view in the directory `/views/default/page/elements/`. In general, this directory contains views that are only used once on a page. This includes views for the sidebar, comment area, and content title along with the topbar, header, and footer. A theme may need to override one or more of these page elements or the page shell to achieve the desired HTML structure.

The structure inside of the `elgg-page-body` `div` is controlled by a layout view. The layout views are found in `/views/default/page/layouts/` and include one, two, and three column layouts. The most commonly used layout is the two column layout called `one_sidebar` and its organization looks like the following image:



Besides page shells, layouts, and page elements, there are also page components. They can appear multiple times on a page and their views are located in the directory `/views/default/page/components/`. We talk about these more in the section on Elgg's CSS framework.

The following is a quick overview of other commonly encountered view directories that contribute to a site's HTML:

- **forms**: the bodies of a site's core forms.
- **input**: input fields such as textboxes, drop downs (selects), checkboxes, and buttons.
- **navigation**: breadcrumbs, menus, tabs, and pagination.
- **object**: display content such as blogs, bookmarks, and files. Mostly in plugins.
- **output**: convenient ways of displaying what was collected from input views.
- **river**: activity stream views.

For a more complete description of Elgg's views with hints on using and theming them, see *Appendix B*, which is a views catalog. The `Theming Sandbox` under the `Developers` section of the administration area is another good resource for learning about the HTML structure and views of Elgg.

CSS

Elgg's CSS is created using the views system. The core views are located in `/views/default/css/`. The primary CSS view is `css/elgg`. It includes the individual theme modules from the elements directory:

- **buttons**: submit, delete, cancel, and action buttons
- **component**s: lists, gallery, image blocks, tables, tags, and river
- **core**: `clearfix` and other special classes
- **forms:** forms and input fields
- **grid**: fluid grid
- **helpers:** utility classes to control spacing, separators, and orientation
- **icons**: icon sprites
- **layout**: page shell and layout classes
- **misc**: CSS that does not fit anywhere else
- **modules**: modules (sidebar boxes, pop-ups) and widgets
- **navigational**: breadcrumbs, menus, pagination, tabs
- **reset**: CSS reset
- **typography**: fonts, headings

There are two important features supported because Elgg uses its views system to create the CSS file: plugins add their CSS by extending the main CSS view and PHP can be used in the CSS. Plugins do not need to include their own CSS file or use inline CSS. Instead, they extend the `css/elgg` view and each time the CSS file is generated, their style information is included. The use of PHP is not very common in Elgg's CSS views, but it is possible to create a color scheme that is controlled through PHP variables. This is especially useful to themers who produce several color schemes of the same theme.

Elgg's CSS is namespaced to prevent conflicts with external CSS files. It also contains PHP comments to explain tricky or non-obvious rules, as follows:

```
<?php // force vertical scroll bar ?>
html, body {
  height: 100%;
  margin-bottom: 1px;
}
```

JavaScript

Like CSS, Elgg's JavaScript is generated with the views system. The primary JavaScript view is `js/elgg`. As with CSS, plugins can extend this view to include additional JavaScript. If a plugin requires a lot of JavaScript or only needs it on certain pages, then it may register a separate JavaScript view with the Elgg engine.

Elgg's JavaScript is modular and the core libraries and classes are in the `/js/` directory. The code is namespaced and encourages plugins to create their own namespaces under the `elgg` space though the `elgg.provide()` function, as follows:

```
elgg.provide('elgg.thewire');

elgg.thewire.init = function() {
  $(".thewire-previous").live('click', elgg.thewire.viewPrevious);
  ...
}

/**
 * Display the previous wire post
 */
elgg.thewire.viewPrevious = function(event) {
  ...
}

elgg.register_hook_handler('init', 'system', elgg.thewire.init);
```

If you have read *Chapters 7* and *8*, then `elgg.register_hook_handler` should look familiar. Elgg's JavaScript framework provides a parallel hook system to the PHP plugin hook system. It enables the engine and plugins to register, trigger, and respond to events that are specific to Elgg.

Besides `js/elgg`, Elgg has two other JavaScript views included on every page. The `js/initialize_elgg` view is included in the HTML `head` element and includes variables that change from page to page. This is kept separate so that `js/elgg` is cacheable. There is also a language JavaScript view `js/languages` that supports internationalization of dynamic content through `elgg.echo()`:

```
var confirmText = elgg.echo('question:areyousure');
```

Graphics

Elgg's graphics are located in `/_graphics/`. The images of primary interest to themers are the avatar icons in `/_graphics/icons/user/` and the icon sprites (`elgg_sprites.png`). There is also an Ajax loading animation available by using the view `graphics/ajax_loader`.

Views system

It is vitally important to understand Elgg's view system when theming. *Chapters 7* and *8* covered views in their tutorials. Here is a condensed review of views in Elgg that can serve as a quick reference.

What is a view?

A view produces a chunk of HTML code (can also produce CSS, JavaScript, XML, and so on). Views are the building blocks of a web page in Elgg. The topbar, sidebar, and footer are all examples of views. Displaying a view works as shown in the following line of code:

```
echo elgg_view('page/elements/footer');
```

This returns a string that contains the HTML for the footer.

Views can contain other views. For example, the header view includes the header logo view.

The output of a view changes based on the variables passed to it, as follows:

```
echo elgg_view('input/text', array('name' => 'email'));
```

The output also varies based on other factors such as who the viewer is. For example, the owner of a blog post sees edit and delete links while a visitor does not.

Extending a view

Using `elgg_extend_view()`, content can be added to the output of a view. The additional content is the output of another view. In this example, the output of the `myplugin/footer` view is appended to the output of `page/elements/footer`:

```
elgg_extend_view('page/elements/footer', 'myplugin/footer');
```

This content can be inserted before the original output or appended after it. See *Lesson 3* in *Chapter 8* for a detailed example of extending a view.

Overriding a view

Plugins can completely replace the output of a view by overriding it. Views get their names based on their location in the directory structure. The `page/elements/footer` view is located at `/views/default/page/elements/footer.php`. When a plugin has a view in the same location as the core view, Elgg will use the plugin's view. Plugin order matters if more than one plugin overrides a view. The last plugin to load takes precedence. This is why it is recommended that a theme plugin be loaded last. For a detailed example of overriding a view, see *Lesson 2* in *Chapter 8*.

Template language

All the major open source web applications have a template system. Templates help to separate the display and layout code from the program logic. They also encourage code reuse. A template consists of tags that are replaced with content by the template engine when a web page is requested.

In Elgg, the view system handles the template processing with PHP as the template language. Elgg creates a page by combining the output of different views (templates):

```
<?php if (elgg_is_logged_in()) { ?>
<div class="elgg-page-topbar">
  <div class="elgg-inner">
    <?php echo elgg_view('page/elements/topbar', $vars); ?>
  </div>
</div>
<?php } ?>
```

Whenever you see a call to `elgg_view()`, think of it as a template tag with parameters. In this example, we are testing whether the viewer is logged in and if so including the template for the topbar.

Caching

Dynamically creating the output of the views takes time, which is why Elgg caches the output of some of the CSS and JavaScript views. Elgg creates a new version of a cached view whenever a plugin is enabled or disabled or when plugin order is changed. When editing a theme, this caching must be turned off. This can be done on the Developer Settings page provided by the developer's plugin.

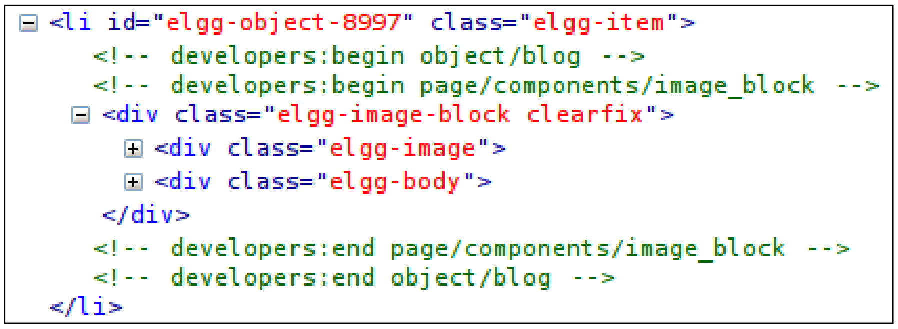
Scanning all the plugins to check for view overrides takes time. Elgg does this every time a page is requested. For better performance, Elgg caches this information. This file view caching should be turned off while doing development.

The viewtype

When you view a blog post in Elgg using your web browser, Elgg creates a web page using HTML. This is done using the **default viewtype**. If you subscribe to the RSS comment feed for the same post, then Elgg uses the **rss viewtype**. In both of those instances the processing and database queries are the same, but Elgg creates different output based on the viewtype. Later, we use a **mobile viewtype** to send web pages formatted for mobile devices in one of our examples.

Tools

One challenge for themers is finding the view that created the HTML that you want to change. The Developer Settings in the administration area have an option for wrapping views in comments. It is then easy to find the view by using a web developer tool such as Firebug:



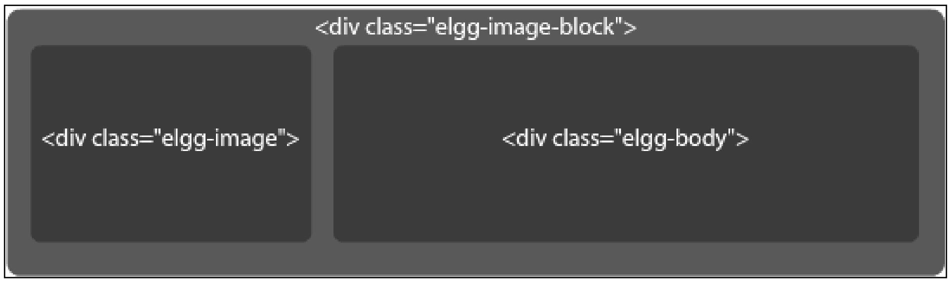
```
<li id="elgg-object-8997" class="elgg-item">
    <!-- developers:begin object/blog -->
    <!-- developers:begin page/components/image_block -->
  <div class="elgg-image-block clearfix">
    <div class="elgg-image">
    <div class="elgg-body">
  </div>
    <!-- developers:end page/components/image_block -->
    <!-- developers:end object/blog -->
</li>
```

CSS framework

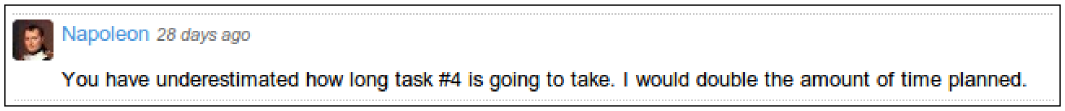
Elgg includes its own CSS framework tailored for social applications. We have already described its modularity. It was also designed to provide reusable CSS classes for plugin authors and reduce the amount of CSS that a themer needs to learn and modify. Some of the principles were borrowed from Nicole Sullivan's OOCSS framework (`https://github.com/stubbornella/oocss`).

Creating abstractions of common visual patterns

There are visual patterns that are seen throughout an Elgg site (and social websites in general). To avoid duplication and encourage reuse of CSS, these patterns have been codified in Elgg's CSS framework as objects. An object consists of HTML and associated CSS that work together to create a building block. An example of one of these objects is the image block, as follows:



This pattern of image on left and text on the right is used throughout the Web from Amazon to eBay, from Facebook to Twitter. An example from Elgg is a comment, as shown in the following screenshot:



Napoleon 28 days ago

You have underestimated how long task #4 is going to take. I would double the amount of time planned.

If a pattern is not identified and codified, then there will be many implementations of it in a site's CSS. As an example, in Elgg 1.7 a sidebar box was reimplemented in different ways by several of the plugins distributed with Elgg.

A best practice for working with CSS objects is to use descendant selectors inside the object, but not across containers and objects. Here is an example of styling across components as this statement affects image blocks that are within a list:

```
.elgg-list-bad > .elgg-image-block {
  background: red;
}
```

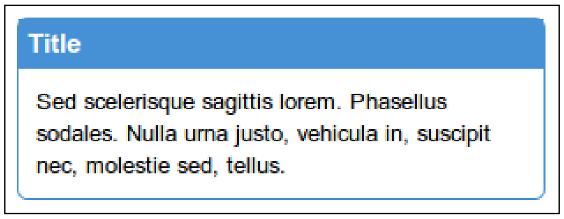
Styling across object boundaries creates dependencies between an object and its container, making the object less predictable. It also increases the possibility of specificity wars.

Customizing objects through extension classes

The CSS objects have a base class that defines what is common across all instances of the object (usually structure). The chrome is added through extension classes. This enables a themer to add a single class to an object to change its overall look. As an example, consider the module object. It consists of a header and body (and optional footer). The HTML for the object looks like the following code snippet:

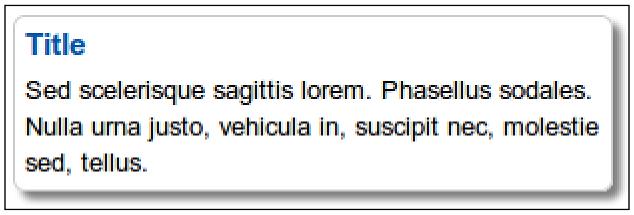
```
<div class="elgg-module">
  <div class="elgg-head">
    <h3>Title</h3>
  </div>
  <div class="elgg-body">
    content here
  </div>
</div>
```

Adding the class `elgg-module-featured` to the top-level `div` results in a module that looks like the following screenshot:


Title

Sed scelerisque sagittis lorem. Phasellus sodales. Nulla urna justo, vehicula in, suscipit nec, molestie sed, tellus.

If we added the class `elgg-module-popup` instead, then we would see the following screenshot:


Title

Sed scelerisque sagittis lorem. Phasellus sodales. Nulla urna justo, vehicula in, suscipit nec, molestie sed, tellus.

The presentation of the module is controlled by the extension classes. This leads to simpler HTML markup and easy reuse and theming of the objects. This technique is used throughout Elgg's CSS framework.

Adding external JavaScript and CSS

Elgg provides functions for managing JavaScript and CSS resources. A plugin registers a JavaScript resource with the Elgg engine like the following code snippet:

```
$js_url = 'mod/pages/vendors/jquery-
  treeview/jquery.treeview.min.js';
elgg_register_js('jquery-treeview', $js_url);
```

This tells the engine the resource exists and gives it an identifier. Any plugin can ask the engine to load the resource on a page with the `elgg_load_js()` function:

```
elgg_load_js('jquery-treeview');
```

There are parallel functions for css: `elgg_register_css()` and `elgg_load_css()`.

If the JavaScript or CSS is being dynamically generated with the views system, then Elgg has a convenience function for determining the URL of the cached view:

```
elgg_register_simplecache_view('js/lightbox');
$lightbox_js_url = elgg_get_simplecache_url('js', 'lightbox');
elgg_register_js('lightbox', $lightbox_js_url);
```

In the preceding code snippet, we register the `js/lightbox` view as cacheable and then register it as a JavaScript resource. Not shown is an optional parameter to specify whether the JavaScript is included in the HTML head element or the footer.

Menu system

Elgg has a unified API for creating, managing, and rendering menus. There are 12 unique menus defined by the Elgg engine and up to 10 of them can appear on a single web page. Modifying the menus is a common activity when building a theme.

Registering a menu item

There are two methods for adding an item to a menu. The first is the `elgg_register_menu_item()` function. *Chapter 7* provides a demonstration of adding items to the site menu and the page menu. The chapter also explains how to use either an `ElggMenuItem` object or an array of parameters with the registration function. In the following example code, a link is added to the sidebar menu whenever we are serving a blog or bookmarks page:

```
elgg_register_menu_item('page', array(
  'name' => 'world',
  'text' => 'Hello world',
  'href' => 'hello/world',
  'contexts' => array('blog', 'bookmarks'),
));
```

The function can be used to create nested menus by passing the name of an item's parent when registering it. This and other parameters are documented in the navigation library found at `/engine/lib/navigation.php`.

The second method adds an item to a menu just before it is rendered. This is used mostly for context-sensitive menus such as a user's hover menu. With this menu, the links depend on the user and so cannot be registered during Elgg's initialization process. Instead, a plugin hook is triggered when a menu has been requested. The plugin hook parameters include the information required to create the link. The following function is from the messages plugin and adds a `Send a message` link to a user's hover menu:

```
function messages_user_hover_menu($hook, $type, $return, $params) {
  $user = $params['entity'];

  if (elgg_get_logged_in_user_guid() != $user->guid) {
    $url = "messages/compose?send_to={$user->guid}";
    $item = new ElggMenuItem('send',
      elgg_echo('messages:sendmessage'), $url);
    $item->setSection('action');
    $return[] = $item;
  }

  return $return;
}
```

For more information on plugin hooks, review *Lesson 8* in *Chapter 8*.

Rendering a menu

A menu is rendered using Elgg's view system by calling `elgg_view_menu()`. This function accepts the name of the menu and an optional array of parameters such as a sorting technique or an additional CSS class for the menu. The `elgg_view_menu()` function uses views located in `/views/default/navigation/menu/`. It selects the view based on the name of the menu. The site menu uses the `navigation/menu/site` view and the page menu `navigation/menu/page`. If there is no view specifically defined for a menu, then `elgg_view_menu()` will use `navigation/menu/default`.

The menu views create unordered lists to hold the menu items. A CSS class is added to the unordered list based on the name of the menu (for example, `elgg-menu-site` for a site menu). The views also handle menu sections, nested menus, and selected menu items. The Theme Sandbox page has examples of the menus' HTML structure and CSS classes.

Comparing theming in WordPress to Elgg

As WordPress is such a popular web application, many people are familiar with creating or editing WordPress themes. For this reason, a comparison between the themes in WordPress and Elgg is useful. If you have not worked with theming in WordPress, then you can skip this section without missing anything.

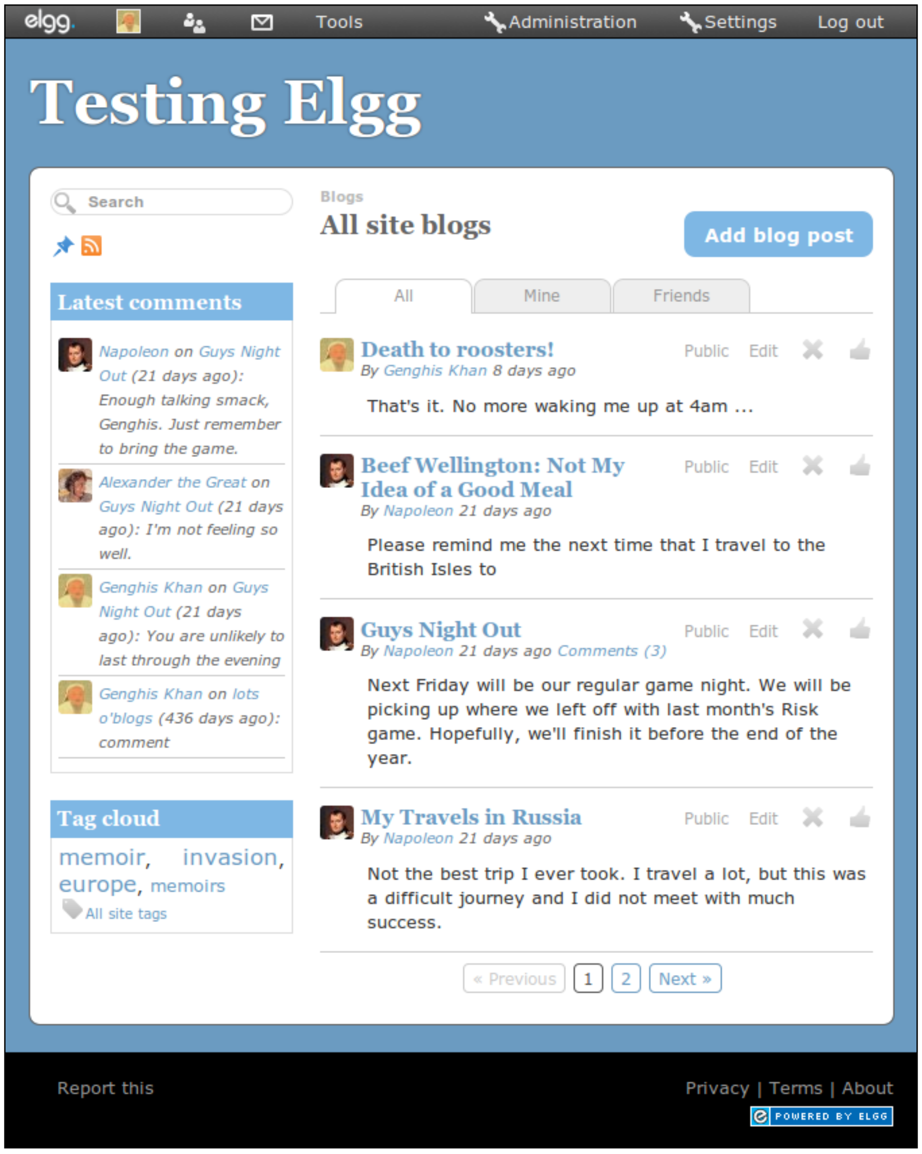
We have already described how themes in Elgg override parts of the default theme. WordPress is different in that you disable the default theme and activate a new one. WordPress and Elgg both use the same template language – that being PHP itself. They obviously have a different set of template functions (`get_header()` for WordPress versus `elgg_view`(`'page/elements/header'`) for Elgg).

A major difference between the two is that with WordPress each type of page has a separate template. There are template files for the main index, archives, a single blog post, and the search page. Each of those template files lays out the HTML code by including templates for the header, footer, sidebar, plus other template tags to include the blog post text and comments. With Elgg, there is a common page shell that lays out the topbar, header, and footer. Most pages in Elgg only control the content in the center of the page.

Another difference is the flexibility gained through using a plugin to create a theme. If you want to imitate Google by using a different header graphic on special occasions, then you do not need to edit your theme plugin. Instead, you can create another plugin that only sets the header. To build on that example, suppose you are a web developer with several clients using Elgg. You could build a base theme that is the same for all of them and provide a secondary theme plugin that includes the customizations for a specific client. This makes development and maintenance of these client sites easier.

Building a theme

We are going to build a theme that puts into practice much of what you learned in the *Theming Basics* section. This tutorial demonstrates how to override Elgg's primary CSS view, how to extend and override views to change the HTML structure of pages, and how to interact with the menu system. We do not cover overriding or adding to Elgg's JavaScript libraries. If this is required for your theme, then read *Lesson 7* in *Chapter 8* for an introduction to adding JavaScript libraries or visit the Elgg wiki.



Testing Elgg

Latest comments

Napoleon on Guys Night Out (21 days ago): Enough talking smack, Genghis. Just remember to bring the game.

Alexander the Great on Guys Night Out (21 days ago): I'm not feeling so well.

Genghis Khan on Guys Night Out (21 days ago): You are unlikely to last through the evening

Genghis Khan on lots o'blogs (436 days ago): comment

Tag cloud

memoir, invasion, europe, memoirs

All site tags

All site blogs

Death to roosters!
By Genghis Khan 8 days ago
That's it. No more waking me up at 4am ...

Beef Wellington: Not My Idea of a Good Meal
By Napoleon 21 days ago
Please remind me the next time that I travel to the British Isles to

Guys Night Out
By Napoleon 21 days ago Comments (3)
Next Friday will be our regular game night. We will be picking up where we left off with last month's Risk game. Hopefully, we'll finish it before the end of the year.

My Travels in Russia
By Napoleon 21 days ago
Not the best trip I ever took. I travel a lot, but this was a difficult journey and I did not meet with much success.

The tutorial is divided into five sections, as follows:

1. **Plugin structure**. This covers creating a theme skeleton and provides an overview of working with the primary CSS view.
2. **Layout**. We demonstrate working with the default page shell HTML to create the layout of the site.
3. **Moving the search box**. This section describes how extending views can be used to change the default structure of a page.
4. **Sidebar box styling.** Theming one of the CSS objects is the focus of this part.
5. **Moving the site menu.** The menu system is used to move the site menu from the page header into the topbar as a drop-down menu.

The theme used in the screenshot includes a few other modifications including styling buttons, tabs, lists, and fonts. They are left out of the tutorial as they require only basic CSS skills and little-to-no particular knowledge of Elgg.

Plugin structure

As a theme is a plugin, it requires a manifest file and a start script. The manifest can be copied from another plugin and modified to describe the theme plugin. The manifest ought to state the plugin's category as *theme* so that it works with the category filter on the Plugins page. The start script can be empty for many themes if the theme does not interact with the menu system or extend any views. We start with an empty `start.php` file and add to it when moving the search box.

To support modifications of wording on the site, we include a languages directory and add an English language file. We are only defining a single language string in the tutorial, but you can read *Lesson 1* of *Chapter 8* for a more detailed example of using language files.

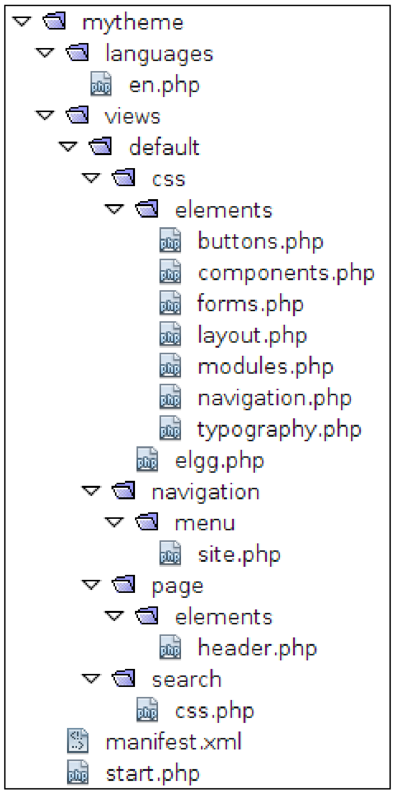
A theme should override Elgg's main CSS view `css/elgg`. This requires putting an `elgg.php` file in the plugin's `/views/default/css/` directory. We have two options on how to proceed with overriding the `css/elgg`:

1. Create a blank view.
2. Copy over Elgg's view into our theme.

The advantage of the first option is that the theme is more likely to look different from other Elgg themes. Themes that start with the default theme often do not have a unique look and feel that set them apart from other Elgg sites. The advantage of copying the original CSS view is that it requires less work because we do not have to theme every single element. In this tutorial, we are copying the original CSS view into ours.

Another choice is whether to follow Elgg's modular CSS structure or place all of the CSS in the `css/elgg` view. This is a matter of personal preference and in this tutorial we use the modular structure. We override seven of the elements views in our theme. If we were replacing the icon set, then we would also override the icons view to define our own sprite.

The final structure is shown in the following image. We describe the addition of the views in the navigation, page, and search directories in their respective sections.



```
mytheme
    languages
        en.php
    views
        default
            css
                elements
                    buttons.php
                    components.php
                    forms.php
                    layout.php
                    modules.php
                    navigation.php
                    typography.php
                elgg.php
            navigation
                menu
                    site.php
            page
                elements
                    header.php
            search
                css.php
    manifest.xml
    start.php
```

Before continuing to the next section, copy the contents of the original Elgg CSS views into the ones that we have created in this theme plugin.

Layout

The layout of the page is set in the aptly-named layout elements view. It has six sections. We are modifying four of them as we are not changing the system messages and topbar.

Default layout

The default layout CSS sets the background of the web page. We want a medium blue background for the page and a black background for the footer, which gives us:

```
body {
  background: #000;
}
.elgg-page {
  background: #6b9bc1;
}
```

Page header

We selected 800-pixel widget for the content of the site. We apply a relative position rule on `elgg-page-header` so that plugins can absolutely position content within the header.

```
.elgg-page-header {
  width: 800px;
  margin: 0 auto;
  position: relative;
  height: 100px;
}
```

Page body layout

The content area has a white background and rounded corners. The rounded corners will not work with versions of Internet Explorer before IE9. We only set a width on the sidebar because the `elgg-body` class creates a spacing filling `div`.

```
.elgg-page-body {
  width: 800px;
  margin: 0 auto;
  background-color: white;
  border: 1px solid #666;
  border-radius: 8px;
```

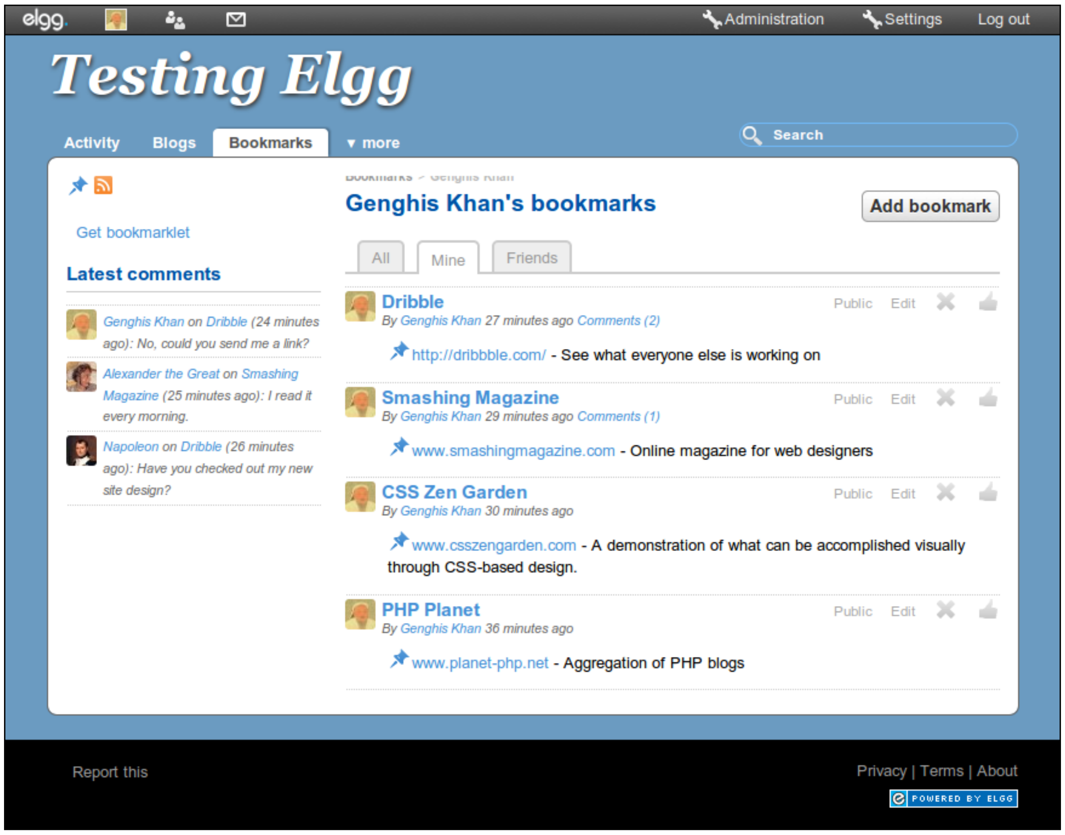
```
    -moz-border-radius: 8px;
    -webkit-border-radius: 8px;
  }
  .elgg-page-body > .elgg-inner {
    margin: 15px;
  }
  .elgg-layout {
    min-height: 360px;
  }
  .elgg-sidebar {
    position: relative;
    float: left;
    width: 210px;
    margin-right: 20px;
  }
  .elgg-main {
    position: relative;
  }
  .elgg-main > .elgg-head {
    margin-bottom: 15px;
  }
```

Page footer

We want the footer to horizontally fill the bottom of the screen yet restrict its content to the central portion of the layout. The inner div allows us to do that.

```
  .elgg-page-footer {
    margin-top: 20px;
    padding-top: 15px;
    background: #000;
  }
  .elgg-page-footer > .elgg-inner {
    position: relative;
    width: 800px;
    margin: 0 auto;
  }
  .elgg-page-footer a:hover {
    color: #ccc;
  }
```

The following screenshot shows how the site appears after these changes:



Genghis Khan's bookmarks

Latest comments

Genghis Khan on Dribble (24 minutes ago): No, could you send me a link?

Alexander the Great on Smashing Magazine (25 minutes ago): I read it every morning.

Napoleon on Dribble (26 minutes ago): Have you checked out my new site design?

Dribble
By Genghis Khan 27 minutes ago Comments (2)
http://dribbble.com/ - See what everyone else is working on

Smashing Magazine
By Genghis Khan 29 minutes ago Comments (1)
www.smashingmagazine.com - Online magazine for web designers

CSS Zen Garden
By Genghis Khan 30 minutes ago
www.csszengarden.com - A demonstration of what can be accomplished visually through CSS-based design.

PHP Planet
By Genghis Khan 36 minutes ago
www.planet-php.net - Aggregation of PHP blogs

Moving the search box

Our next objective is to move the search box from the page header to the sidebar. This requires adding an initialization function to our plugin's start script:

```
elgg_register_event_handler('init', 'system', 'mytheme_init');

function mytheme_init() {
  // search box in sidebar
  elgg_unextend_view('page/elements/header', 'search/header');
  elgg_extend_view('page/elements/sidebar', 'search/search_box',
    100);
}
```

Originally, the search header extended the page header view so we unregister it. Then, we extend the sidebar with the search box and set the priority at 100. Using a priority less than 500 causes the search box to be prepended to the sidebar rather than appended.

We also need to theme the search box. The original theming was in the search plugin in the `search/css` view. Because of Elgg's CSS framework, well written plugins should have very little CSS. The search plugin is an exception to this because it styles the search box and sets colors for highlighting search terms.

We copy the original CSS in our plugin's `search/css` view and replace the search box CSS with the following:

```
.elgg-sidebar .elgg-search {
  margin-bottom: 15px;
}

.elgg-sidebar .elgg-search input[type=submit] {
  display: none;
}
.elgg-search input[type=text] {
  -webkit-border-radius: 10px;
  -moz-border-radius: 10px;
  border-radius: 10px;

  border: 1px solid #ddd;
  color: #999;
  font-size: 12px;
  font-weight: bold;
  padding: 2px 4px 2px 26px;
  background: transparent url(<?php echo elgg_get_site_url(); ?>_
graphics/elgg_sprites.png) no-repeat 2px -934px;
}
```

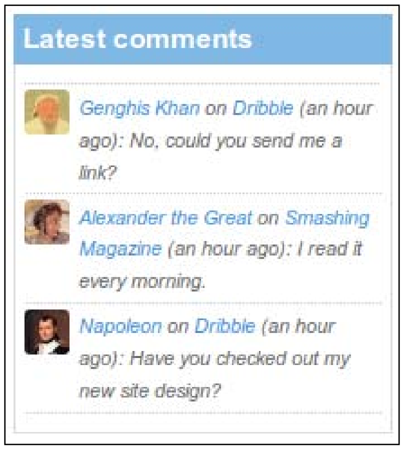
This CSS makes a few adjustments to its position and colors.

Styling the sidebar module

As an example of using a CSS extension class, we style the `elgg-module-aside` class that is commonly used in the sidebar. Find this class in the modules elements view and replace its rules with the following:

```
.elgg-module-aside > .elgg-head {
  background: #7eb7e4;
  padding: 5px;
}
.elgg-module-aside > .elgg-head * {
  color: white;
}
.elgg-module-aside > .elgg-body {
  border: 1px solid #ddd;
  border-top: 0;
  padding: 5px;
}
```

The module now looks like the following screenshot:


Latest comments

Genghis Khan on Dribble (an hour ago): No, could you send me a link?

Alexander the Great on Smashing Magazine (an hour ago): I read it every morning.

Napoleon on Dribble (an hour ago): Have you checked out my new site design?

Moving the site menu to the topbar

To unify site navigation, we move the site menu into the topbar as a drop-down menu. We do this by registering the site menu as an item in the topbar menu. The best method to add the site menu to the topbar is using the `'register'`, `'menu:topbar'` plugin hook. This allows plugins as much time as possible to register their items for the site menu. This is done in the plugin's start script like the following code snippet:

```
elgg_register_event_handler('init', 'system', 'mytheme_init');

function mytheme_init() {
  // tools menu
  elgg_unregister_plugin_hook_handler('prepare', 'menu:site',
    'elgg_site_menu_setup');
  elgg_register_plugin_hook_handler('register', 'menu:topbar',
    'mytheme_menu_setup');
  // search box in sidebar
  elgg_unextend_view('page/elements/header', 'search/header');
  elgg_extend_view('page/elements/sidebar', 'search/search_box', 100);
}

function mytheme_menu_setup($hook, $type, $values) {
  $site_menu = elgg_view_menu('site');

  $item = new ElggMenuItem('tools', $site_menu, false);
  // the tools menu item should be last in default section
  $item->setPriority(1000);
  $values[] = $item;

  return $values;
}
```

In addition to registering for the menu register plugin hook, we also unregistered a handler for a menu prepare hook. The Elgg engine provides this function to handle custom ordering and creating the *more* section of the default site menu.

The site menu is now a part of the topbar menu, but it is also being included in the header. To remove it, we need to override the page header view. To do this, we create the view `page/elements/header` in our plugin, copy the original contents into our view, and remove the call to `elgg_view_menu()`.

We are calling the menu item `Tools` as it has a list of the major tools of the site. To add text for our menu item to the topbar and control the HTML of the drop-down menu, we override the `navigation/menu/site` view. Our new code will be as follows:

```
<?php
/**
 * Turn site menu into drop down menu
 */

$label = elgg_echo('mytheme:tools');
echo "<a>$label</a>";

echo '<ul class="elgg-menu elgg-menu-tools">';
foreach ($vars['menu']['default'] as $menu_item) {
  echo elgg_view('navigation/menu/elements/item', array('item' =>
    $menu_item));
}

echo '</ul>';
```

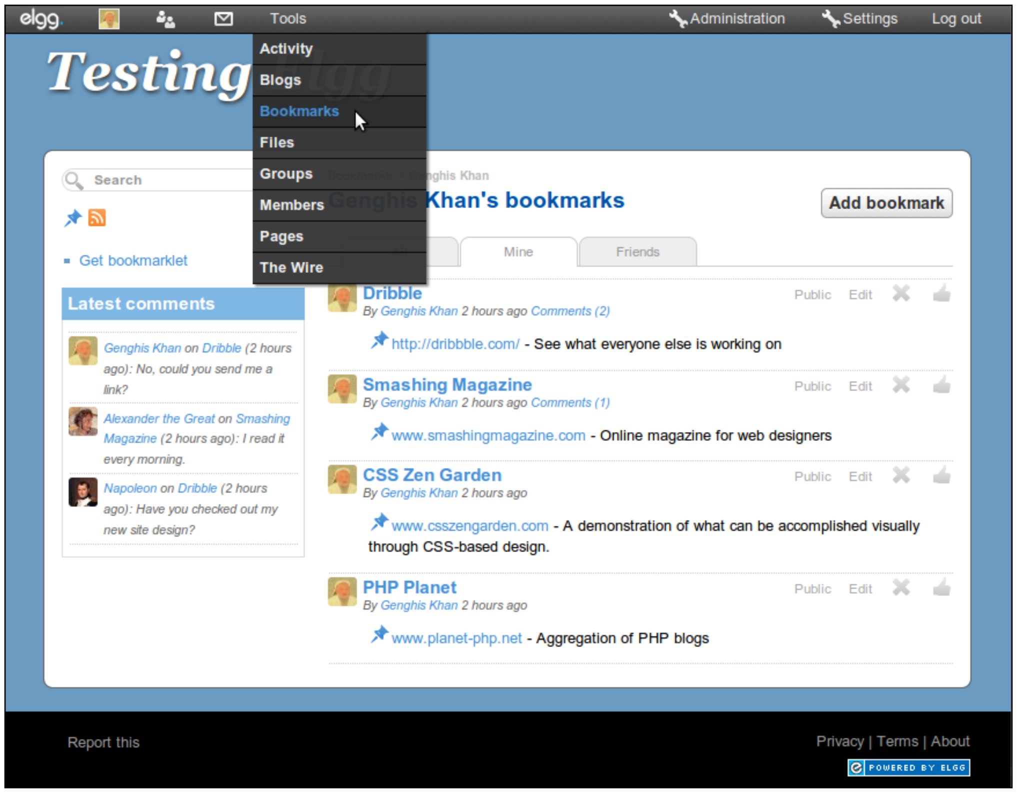
This creates an unordered list with an extension class of `elgg-menu-tools` for theming purposes. The label is added first to provide an element to trigger the drop-down menu. We define the string in the language file:

```
<?php

$english = array(
  'mytheme:tools' => 'Tools',
);

add_translation('en', $english);
```

The final step is adding the CSS to create the drop-down menu. The unordered list is hidden until a user mouses over the `Tools` topbar list element to reveal it. We complete the drop-down menu by styling its links and giving them a hover effect.

```
.elgg-menu-tools {
  z-index: 10;
  display: none;
  position: absolute;
  padding-top: 4px;
  width: 150px;

  -webkit-box-shadow: 1px 1px 1px rgba(0, 0, 0, 0.25);
  -moz-box-shadow: 2px 2px 2px rgba(0, 0, 0, 0.5);
  box-shadow: 1px 1px 1px rgba(0, 0, 0, 0.25);
  opacity: 0.95;
}

li:hover > .elgg-menu-tools {
  display: block;
}
.elgg-menu-tools a {
  background-color: #333;
  color: #eee;
  border-bottom: 1px solid #000;
  font-weight: bold;
  height: 22px;
  padding-bottom: 0;
  padding-left: 6px;
  padding-top: 4px;
}
.elgg-menu-tools a:hover {
  text-decoration: none;
  color: #4690D6;
  background-color: #2a2a2a;
}
```

The end result looks like the following screenshot:



Genghis Khan on Dribble (2 hours ago): No, could you send me a link?

Alexander the Great on Smashing Magazine (2 hours ago): I read it every morning.

Napoleon on Dribble (2 hours ago): Have you checked out my new site design?

Dribble
By Genghis Khan 2 hours ago Comments (2)
http://dribbble.com/ - See what everyone else is working on

Smashing Magazine
By Genghis Khan 2 hours ago Comments (1)
www.smashingmagazine.com - Online magazine for web designers

CSS Zen Garden
By Genghis Khan 2 hours ago
www.csszengarden.com - A demonstration of what can be accomplished visually through CSS-based design.

PHP Planet
By Genghis Khan 2 hours ago
www.planet-php.net - Aggregation of PHP blogs

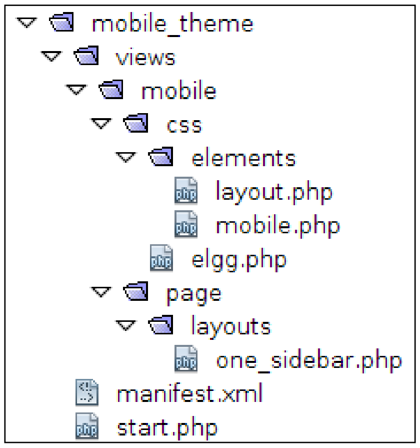
This leaves the typography, form elements, buttons, and the other menus to be updated by you for this theme. The available code for this tutorial has one possible implementation of this design, but doing it yourself is a good exercise to learn more about theming Elgg. The code can be downloaded at `http://www.packtpub.com/elgg-18-social-networking/book`.

Creating a mobile theme

We close the chapter with a short guide to creating a mobile theme. Elgg's view system makes it easy to create a theme for mobile devices. A mobile theme creates another viewtype that is used whenever a mobile browser is detected. In *Chapter 7*, we discussed how Elgg uses a default viewtype to serve web pages to a browser and an rss viewtype to serve RSS feeds to a feed aggregator. The logic and data are the same for different viewtypes, but the rendered output is different. For a mobile theme, we create a mobile viewtype and switch to this viewtype whenever a mobile browser is detected.

Plugin structure

Our mobile theme is structured like the preceding theme except that instead of putting the views in `views/default/`, they are added to `views/mobile/`, as follows:



In the start script, we detect mobile browsers by analyzing the user agent of the request. There are toolkits available to do this for you such as **WURFL** (`http://wurfl.sourceforge.net/nphp/`) and `http://detectmobilebrowser.com/`. We leave it up to you to select one, or code the plugin to always serve the mobile theme. The code in the start script looks like the following code snippet:

```
<?php
/**
 * Mobile theme
 *
 * An example of detecting mobile browsers and changing the viewtype
 */

elgg_register_event_handler('init', 'system', 'mobile_theme_init');

function mobile_theme_init() {
  elgg_register_viewtype_fallback('mobile');

  mobile_theme_set_viewtype();

  // do not want the more menu
  elgg_unregister_plugin_hook_handler('prepare', 'menu:site', 'elgg_
site_menu_setup');
}
```

```
function mobile_theme_set_viewtype() {

  $user_agent = $_SERVER['HTTP_USER_AGENT'];

  // your code here
  elgg_set_viewtype('mobile');
}
```

In the initialization function, we register our mobile viewtype as one that falls back to default. Doing this causes Elgg to use a view from the default viewtype if one does not exist in the mobile viewtype. That way we do not need to create a mobile view for every single default view, but only for the ones that need to be structured differently for mobile browsers.

The `mobile_theme_set_viewtype()` function checks the mobile devices based on the user agent string sent by the browser. If it detects a mobile device, then it switches Elgg into the mobile viewtype. Of course, in the preceding code it is always set to the mobile viewtype.

Layout

The only change we are making to the layout is removing the sidebar. We do this by adding a `page/layouts/one_sidebar` view in the mobile viewtype. After we copy the code from the original view and remove the section for the sidebar, we now have a single column site.

CSS

It is best that a mobile theme have a fluid layout to accommodate smaller screen resolutions. We convert the default theme to a fluid theme by overriding the layout CSS view. This is accomplished by removing the fixed widths on `div`s. When we are done, the layout CSS is pared down to the following:

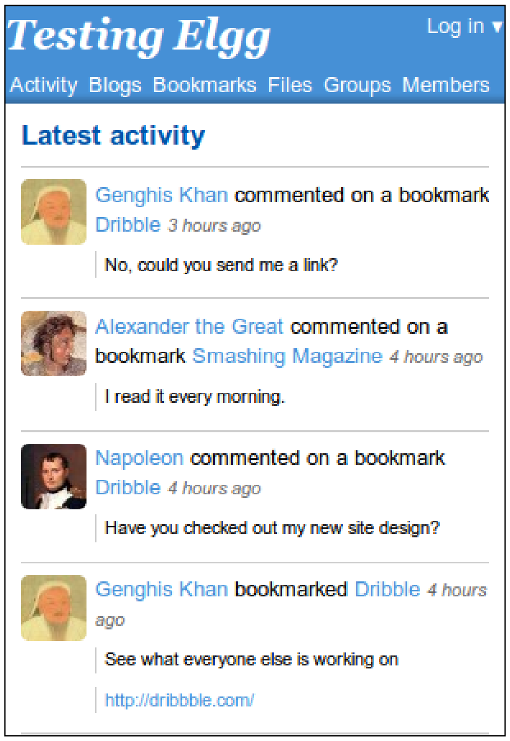
```
/***** TOPBAR ******/
.elgg-page-topbar {
  background: #333333 url(<?php echo elgg_get_site_url(); ?>_graphics/
toptoolbar_background.gif) repeat-x top left;
  border-bottom: 1px solid #000000;
  position: relative;
  height: 24px;
  z-index: 9000;
}
```

```
/***** PAGE MESSAGES ******/
.elgg-system-messages {
  position: fixed;
  top: 24px;
  right: 20px;
  z-index: 2000;
}
.elgg-system-messages li {
  margin-top: 10px;
}
.elgg-system-messages li p {
  margin: 0;
}

/***** PAGE HEADER ******/
.elgg-page-header {
  position: relative;
  background: #4690D6 url(<?php echo elgg_get_site_url();
  ?>_graphics/header_shadow.png) repeat-x bottom left;
}
.elgg-page-header > .elgg-inner {
  position: relative;
  height: 60px;
}

/***** PAGE BODY LAYOUT ******/
.elgg-layout {
  min-height: 360px;
}
.elgg-main {
  position: relative;
  padding: 10px;
}
.elgg-main > .elgg-head {
  padding-bottom: 3px;
  border-bottom: 1px solid #CCCCCC;
  margin-bottom: 10px;
}

/***** PAGE FOOTER ******/
.elgg-page-footer {
  position: relative;
}
.elgg-page-footer {
  color: #999;
}
.elgg-page-footer a:hover {
  color: #666;
}
```

We also override the primary CSS view, copy the original code into it, and add a line near the bottom to include a new mobile CSS view.

```
  echo elgg_view('css/elements/mobile', $vars);
```

This mobile CSS view is used to override specific elements of the default theme without overriding complete views. We decrease the font sizes and simplify the site menu. When we are done, we have a site that looks like the following screenshot:



Latest activity

Genghis Khan commented on a bookmark Dribble 3 hours ago

No, could you send me a link?

Alexander the Great commented on a bookmark Smashing Magazine 4 hours ago

I read it every morning.

Napoleon commented on a bookmark Dribble 4 hours ago

Have you checked out my new site design?

Genghis Khan bookmarked Dribble 4 hours ago

See what everyone else is working on

http://dribbble.com/

Because much of Elgg's CSS uses fluid layouts, there does not need to be much restyling of the content. Most of the work in creating a mobile theme is with navigation. For example, we did not theme the topbar. It takes up too much space without changes for a mobile display.

Summary

This chapter explained the basics of building a theme and included a tutorial for creating one. Theming pulls together many skills: graphics design, CSS and HTML, client-side programming with JavaScript, and working with Elgg's view system. With the right skills, you can create impressive websites with Elgg that draw viewers in and keep them coming back.

In the next and final chapter of this book, we walk through the steps of setting up and running a production website using Elgg.

10
Moving to Production

This chapter describes what is needed to move from building and testing your website to using a production server. The server that you used for testing Elgg may be a laptop, an inexpensive shared hosting account, or the intended production server. Up to this point, you have been installing and removing plugins, creating a theme, and trying out Elgg's capabilities without concern for maintaining a stable site. Backups, server performance, and spammers were also not concerns during the early phase of the project.

Now you are planning the production site and have a lot of questions. What sort of hosting plan do I need? How can I get the most out of my server? How do I back up my site? What do I do if e-mails from the site are being marked as spam? How do I upgrade to the latest version of Elgg? This chapter answers these types of questions. It is a basic introduction to running a production server.

Not all of these questions are relevant to every user of Elgg. If you are supporting a fixed user population, then growth will not be an issue. If you have an administrator to maintain the server, then you will not be responsible for several of the areas covered. This chapter serves as a guide as problems and questions arise and will, at a minimum, point you in the right direction.

This chapter covers the following topics:

- Selecting a server or choosing a hosting provider
- Configuring a server
- Backing up an Elgg site
- Dealing with spammers
- Testing and upgrading Elgg
- Improving performance
- Migrating to a new server

Selecting a server

Selecting a server is a very important step in creating a successful site. It can be the difference between a slow site that people quickly abandon and a site with snappy performance that people enjoy using. This section provides an overview of the selection process and includes a discussion of different types of hosting services.

Performance considerations

There is no standard server or hosting package that works for every Elgg site. Performance benchmarking data captured during operation of a production site is the best information for selecting a server. Unfortunately, that type of data is not available when launching a new site. The next best approach is to estimate what you think you need and plan to upgrade if your site requires more resources. This section provides background information to help you understand your options.

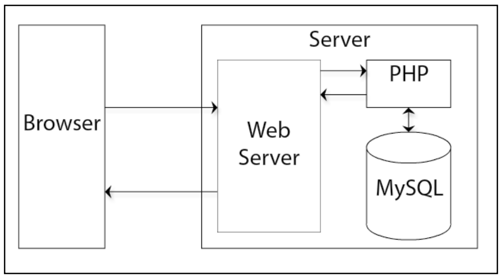
Competing for resources

There are five limited resources that every web application needs: network bandwidth, processing power, memory, file I/O, and disk space.

1. **Network bandwidth**: the amount of data that can be transmitted by the server over some period of time.
2. **Processing power**: the number of computations the server can perform over a period of time. The number and speed of the server's CPUs determine how much processing power a server has.
3. **Memory**: the amount of **Random Access Memory** (**RAM**).
4. **File I/O**: file input/output - the amount of data that can be read from and written to the file system over some period of time.
5. **Disk space**: the amount of storage space on the server's hard drives.

With Elgg, there are three primary processes running on a server: the web server, PHP, and the database. (PHP often runs within the web server process, but for this discussion we will keep them separate.) The three processes compete for CPU time, memory, and file I/O.

When a web browser requests a page from the server, the request is turned over to the web server. It in turn gives the request to PHP which pulls information from the database. PHP uses this data to create a web page which it gives to the web server for transmission to the browser. When the browser receives the web page, it then requests the images, JavaScript, and CSS files linked to in the web page. These requests are also processed by the web server and all contribute to the load on the server. This communication process is depicted in the following diagram:



As the amount of traffic to the site increases, one of those limited resources will become exhausted and the performance of the website will degrade. The resource that runs out first is called the bottleneck. Part of server selection is balancing the resources so that the server does not run out of RAM quickly while having a lot of CPU cycles free.

Usage patterns

When estimating what server resources are needed, the place to start is creating a profile of expected activity on the site. The total number of users rarely matters, but the number of concurrent users does. Every view of a web page results in several files being requested from the server, as mentioned earlier, HTML, JavaScript, CSS, and images. Each of those requests keeps a web server process busy for some amount of time. Let's assume that a web browser loads four files at a time from the server and each web server process consumes 64 MB. If there are eight users requesting web pages at the same time, then it quickly adds up:

4 requests × 64 MB × 8 users = 2 GB.

This does not include the memory that the database server is using nor the other background processes of the server. If you are planning for a large number of concurrent users, the amount of RAM and processing power required will be large.

The type of activity matters. Users that upload photos and videos consume significantly more resources than users that write blog posts. A file upload not only leads to a web server process running for a long time to accept the file, but it also consumes large amounts of bandwidth and disk space. As another example, consider users who set their RSS feed readers to check your site's RSS feeds every five minutes. Each check can involve a web server process plus PHP and database calls. This creates a consistent load on the server in addition to the more transient load from people visiting web pages.

General guidelines for server selection

All guidelines are dependent on usage patterns. Heavy use of streaming video, chat, Ajax refreshes, or large file uploads affects what resources could become the bottleneck.

- **The bottleneck for Elgg sites tends to be processor usage first and RAM second**. If you are looking at dedicated servers, then select one with multiple fast cores and plenty of RAM.
- **Leave room to grow**. You do not want your very fast site becoming sluggish after only a month of growth. Once you are in production conditions, check the RAM and processor usage.
- **Develop estimates but do not put too much faith in them**. You should be able to work from an estimate of concurrent users to the amount of required RAM based on how much memory the web server and database use on a test server. It is very difficult, though, to estimate load without observing actual users so treat those estimates as a ballpark figure.
- **Consider average and peak usage**. You do not want a server that works well for the majority of the time, but grinds to a halt during peak-time periods.

Hosting options

A popular option for serving a website is paying a company to host it. These companies purchase the hardware, maintain the servers, perform upgrades, and provide technical support. Hosting options are divided into categories based on how many people share the same server, how much control is given to the users, and the amount of resources available with an account (bandwidth, disk space, RAM, and processors).

Shared hosting

Shared hosting is the most inexpensive hosting option. With a shared hosting account, your site runs on the same server as tens, hundreds, or even thousands of websites. This has ramifications for performance, security, and control, as follows:

- Performance depends on the load created by the other sites. As traffic on other sites spikes, your site becomes slower.
- There is an increased security risk compared to other forms of hosting. Depending on the server configuration, insecure web applications being run by other users can be used by attackers to affect your site.
- Web server configuration and software versions are controlled by the hosting company as any change affects all the sites on the server. In addition, users of shared hosting are limited to web-based configuration programs to set the available server parameters.

Shared hosting packages are generally divided into tiers based on bandwidth and disk space. Exceeding those limits will typically result in suspension of the account.

Shared hosting is not appropriate for most production Elgg sites due to its resource constraints. Use it for testing or very low traffic sites.

Virtual Private Server

A **Virtual Private Server** (**VPS**) is a step up from shared hosting. There are still multiple sites running on the same physical server, but each site gets its own virtual machine. The virtual machines have RAM and disk space allocated to them, but share the CPUs of the physical server. Many hosting companies offer tiers of service along with increased amounts of RAM and disk space. They may also offer different price points based on the number of virtual machines running on a server.

Performance on a VPS depends on the load created by the other users, but there are usually fewer VPS accounts on a physical server than with shared hosting. A VPS provides shell access and can be configured like a physical server. There is often a web-based control panel for those who are not comfortable with editing configuration files. In addition to offering more control than shared hosting, a VPS is more secure by nature of being a virtual machine.

A VPS is a good introduction to paid hosting. It offers better performance and control than shared hosting at a much lower cost than a dedicated server.

Dedicated server

With a dedicated server, you are paying the hosting provider to maintain a physical server for you. It costs much more than shared hosting or a VPS, but you do not have to share the server with anyone else. A dedicated server provides significantly better performance than shared hosting or a VPS.

Dedicated servers are available as managed or unmanaged. With a managed server, the hosting company monitors the server, installs security patches, and performs upgrades. With an unmanaged dedicated server, the maintenance is done by you. Managed servers obviously cost more because of this.

Even with a dedicated server, there are often caps on how much bandwidth can be consumed per month. If your site has extremely high bandwidth requirements, then look into an unmetered server. Besides selecting between managed/unmanaged and metered/unmetered, the hardware configuration determines the price of the service. There are choices on the type of processor, number of processors, amount of RAM, and hard disk size.

A dedicated server is a good choice for a high traffic Elgg site.

Cloud hosting

Cloud hosting is the latest buzzword in hosting circles and what is meant by the term depends on the hosting provider. The general idea is that you are not limited to running on a single physical server, but instead your application runs in the "cloud". The cloud is simply a set of connected servers operated by the hosting company. The promised benefits of cloud hosting are inherent scalability and robustness to hardware failures. The reality is that cloud hosting is not the panacea that is sometimes advertised. Hardware failures can still bring down sites and achieving the advertised scalability can require architectural changes to the web application being hosted.

Depending on the provider, a cloud hosting account can resemble a shared hosting site with limited access or a VPS. Performance depends on how many people are running on the same physical server and the specifications of the server. Do not select cloud hosting just because of the buzz surrounding it. Compare it against VPS and dedicated servers if you are interesting in using this type of hosting.

Hosting company selection

When evaluating a particular hosting company, you should check their reputation in online forums (while realizing that there are no perfect hosting providers out there). Besides looking at reputation, cost, and the technical specifications of hosting packages, there are other questions to consider when selecting a hosting provider.

Does the company provide an upgrade path? As your site grows, you may need additional server resources, whether upgrading a VPS account or moving to a dedicated server. Does the company offer a tier of packages? Would you need to reinstall your site to upgrade?

Is a backup plan available? Hosting providers may back up your data for free, charge an additional fee, or not offer it at all.

How and when is support offered? It is one thing to promise 24/7 support and another to offer easy access to qualified technical support staff. Is support offered by e-mail, chat, or phone? What are the expected wait times? How much does support cost?

What happens if the bandwidth limits are exceeded? Are you notified as you approach the limits? Is your account suspended? What is the cost to purchase additional bandwidth?

Where is the server located? If you expect your users to come from one area of the world, then you can sometimes achieve faster response times by using a server that is physically close to the users. There can also be legal ramifications connected to the location of the server.

Configuring a server

Server configuration covers a wide range of topics (ranging from security to performance to maintenance) and it would take a library of books to cover all of them. In this section, we describe the different server applications that can be configured and tuned. References are provided to books and websites for further reading. Always test configuration changes on a test server before modifying a production server. A faulty configuration can cause the server applications to cease working.

Apache

Apache, the most popular web server software, is likely running on your server. Other web server software includes Internet Information Services (IIS) from Microsoft, lighttpd, and nginx. With a typical Apache installation, configuration is stored in two or three different locations. There is a server-wide configuration file (often named `apache2.conf` or `httpd.conf`). This file includes other configuration files that are specific to a particular site, web application, or module. On a Linux server, these files are located somewhere under the `/etc` directory. The third type of configuration file is the `.htaccess` file. It is included in the same directory as the web content. Its configuration options override any of the previous settings for that directory and its subdirectories.

The standard configuration provided with Apache should work for Elgg with two exceptions. First, Elgg requires that the URL rewriting module be enabled (called `mod_rewrite`). Second, Elgg stores its rewrite rules in its `.htaccess` file which will not be processed unless `AllowOverride` is enabled for its directory. If you need assistance making these configuration changes, then ask the technical support staff for your server or use the many resources available on the web including the Elgg wiki (`http://docs.elgg.org/`).

There is a tremendous amount of information available in Apache's logs. The logging is used to track site traffic, record errors, and debug configuration issues. The standard Apache installation uses two types of logs: access logs and error logs. The access log is written to whenever a request is made of the web server. A log entry will generally contain the address of the requested resource, the time and date of the request, the IP address of the requester, and a web browser description. Not surprisingly, the error log is a record of errors and warnings such as when a web browser requests a page that does not exist.

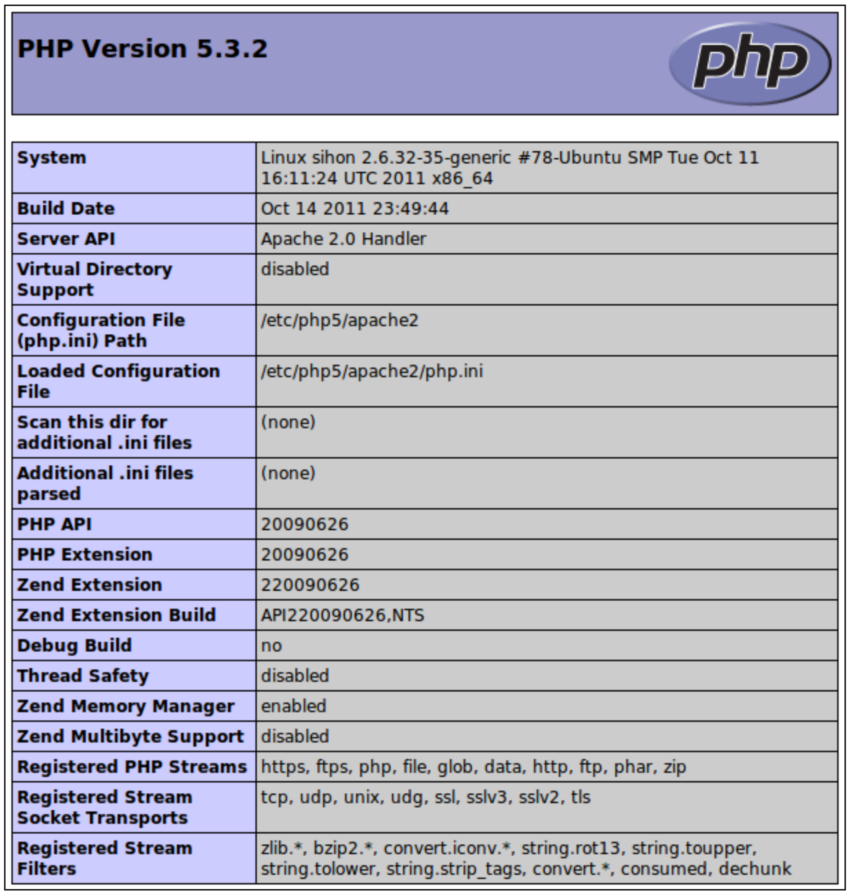
Two valuable online references for working with Apache are the online docs at `http://httpd.apache.org/docs/2.2/` and the Apache wiki at `http://wiki.apache.org/httpd/`. If you prefer a book on the topic, then there are many to choose from. It is best to select one written for your skill level. Beyond the introductory and general guides to the Apache HTTP server, there are books written on security, performance, and development.

PHP

PHP creates the web pages that Apache serves. The primary configuration is stored in a `php.ini` file whose location is server dependent. You can locate this configuration file and take a look at PHP's current configuration by creating a PHP file in your web server's root directory (where your web application code is located) with the code:

```
<?php  phpinfo();  ?>
```

Viewing this page displays the server's PHP configuration:



PHP Version 5.3.2

System	Linux sihon 2.6.32-35-generic #78-Ubuntu SMP Tue Oct 11 16:11:24 UTC 2011 x86_64
Build Date	Oct 14 2011 23:49:44
Server API	Apache 2.0 Handler
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php5/apache2
Loaded Configuration File	/etc/php5/apache2/php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20090626
PHP Extension	20090626
Zend Extension	220090626
Zend Extension Build	API220090626,NTS
Debug Build	no
Thread Safety	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
Registered PHP Streams	https, ftps, php, file, glob, data, http, ftp, phar, zip
Registered Stream Socket Transports	tcp, udp, unix, udg, ssl, sslv3, sslv2, tls
Registered Stream Filters	zlib.*, bzip2.*, convert.iconv.*, string.rot13, string.toupper, string.tolower, string.strip_tags, convert.*, consumed, dechunk

In addition to the options in the `php.ini` file, parameters can also be overridden in `.htaccess` files. Elgg's `.htaccess` file sets several options. Your `php.ini` configuration file probably has over one hundred settings. There are only a small number that ever need to be changed from the defaults. One parameter that Elgg sets in its `.htaccess` file is `memory_limit`. It determines the amount of available memory for processing a PHP script. The PHP code and variables consume this memory. If a script exceeds the limit, then PHP stops with a fatal error. You may need to increase this limit if you are using a large number of plugins. A complete list of core PHP parameters is found at `http://php.net/manual/en/ini.core.php`.

The parameters that you are most likely to adjust are those that deal with memory, execution time, file uploads, session length, and modules. Modules are like plugins for PHP and add significant functionality. Elgg requires that the GD module be enabled as it is used for resizing images. Other modules may be required depending on what Elgg plugins you are running. The best reference material for PHP configuration is on the web. The PHP site (`http://php.net`) and Google searches will find most of the information that you need.

PHP logging is very useful for debugging configuration and application issues. By default the messages are sent to the Apache error log. This can be changed with the `error_log` setting:

```
error_log = /var/log/php.log
```

On a production server, errors should not be written to the screen. This is set with the `display_error` setting. Elgg's `.htaccess` turns the display of errors off by default. The verbosity of the log is determined by `error_reporting`. To report all errors and warnings, the following settings can be used in the `php.ini` file:

```
error_reporting  =  E_ALL & ~E_NOTICE & ~E_STRICT
```

The other important setting is `log_errors`. It should be set to `1`, otherwise fatal errors will not be written to the error log.

MySQL

MySQL is the database server that Elgg uses. Its primary configuration file is named `my.cnf` and is generally located in `/etc` or `/etc/mysql/` on a Linux server. It is rare that an individual new to web applications would need to adjust MySQL's configuration.

MySQL provides three different types of logging: errors, general queries, and slow queries. The error log includes information about the server starting, stopping, and any errors that occur while running. General query logs are typically turned off as they grow too quickly. The slow query log is useful if you are experiencing performance issues and are knowledgeable about SQL.

The MySQL reference is available online at `http://dev.mysql.com/doc/`. There are also general guides to MySQL Administration and books on specific topics such as security and performance.

Cron

Cron is a program that executes tasks at certain times. For example, cron can run a backup script every night at midnight. Elgg uses cron to hit particular web pages at set times. When the web pages are loaded, Elgg runs tasks such as cleaning up garbage in the database or rotating the log. The following is what Elgg's example cron configuration looks like:

```
# Location of GET (see: http://docs.elgg.org/wiki/What_is_get)
GET='/usr/bin/GET'

# Location of your site (don't forget the trailing slash!)
ELGG='http://www.example.com/'

# The crontab
# Don't edit below this line!
@reboot $GET ${ELGG}cron/reboot/
* * * * * $GET ${ELGG}cron/minute/
*/5 * * * * $GET ${ELGG}cron/fiveminute/
15,30,45,59 * * * * $GET ${ELGG}cron/fifteenmin/
30,59 * * * * $GET ${ELGG}cron/halfhour/
@hourly $GET ${ELGG}cron/hourly/
@daily $GET ${ELGG}cron/daily/
@weekly $GET ${ELGG}cron/weekly/
@monthly $GET ${ELGG}cron/monthly/
@yearly $GET ${ELGG}cron/yearly/
```

The `@hourly` line causes `GET` to load the page `http://www.example.com/cron/hourly/` once an hour. When that happens, Elgg notifies any plugins that are registered to run hourly and they do whatever they need to do. The same thing occurs for the other time intervals shown in the preceding example.

Shared hosting sites and some VPSs provide a web interface to set up cron jobs. With a shell account, you can configure cron with the command:

```
crontab -e
```

Running this command loads the default editor with the current **crontab** (cron table). Editing and saving the file reloads the cron jobs. To verify that this is working, check the access log of Apache. It should contain requests for `/cron/minute/` each minute.

E-mail

E-mail plays two vital roles in Elgg: new users receive e-mails to validate their accounts and notification e-mails are sent when activity occurs. Elgg uses the PHP function `mail()` to send e-mails. The `mail()` function passes the message to the server's **Mail Transfer Agent** (**MTA**). The MTA sends the message out on the Internet to start its journey to the recipient's e-mail account.

If you are using a hosting company, then it handles the configuration of PHP and the MTA. Otherwise, you can read more on PHP's mail configuration at `http://www.php.net/manual/en/mail.configuration.php`. Most popular MTAs (such as **Sendmail** and **Postfix**) also have books available for administering them.

Properly configuring PHP and the MTA does not guarantee that your e-mail ends up in your recipients' inbox. The e-mail must pass through spam filters to reach its destination. It is not uncommon for Elgg account validation e-mails to be filtered out by large mail sites such as Yahoo and Hotmail. This typically happens because the mail providers do not trust your server.

There are steps that you can take to increase the probability that your e-mails pass these spam filters. These steps have names such as **Sender Policy Framework** (**SPF**), **DomainKeys**, and **SenderID**. They are authentication protocols that attempt to filter out spoofed e-mails. A spoofed e-mail message is one that claims to be from a particular server, but was actually sent from a different one. These protocols provide a method for checking that the server that sent the message was approved for sending e-mail for that e-mail address. Your hosting provider should be able to perform this configuration for you so that Yahoo and Hotmail will trust your server.

Managing the site

After the server is configured, Elgg is installed, and your site's membership is growing, there are other administrative tasks that should be performed on a daily basis such as backing up your site, stopping spammers, and interacting with the users. This section provides an overview of these important tasks.

Backup

Servers crash and hard drives fail. It is a reality of server administration that cannot be escaped. You need to prepare by regularly backing up your site. There are three primary components to every Elgg site:

1. Code (Elgg and your plugins).
2. Database.
3. User files in data directory.

With backups of these three components, you will always be able to restore a site if a failure occurs. The other components that should be backed up are a server's configuration files (Apache, PHP, MySQL, cron, e-mail server).

Code

You have control of when the code of your site changes, making this back-up task easier. Modifications of the code probably occur rarely for your site so a manual back-up strategy is sufficient. A common approach is creating an archive (zip file or tarball) of the code after every change and saving that on a separate computer. For critical Elgg sites, you should consider using a code repository tool such as Subversion to manage the version of code deployed to the production server. A code repository does more than allow you to restore the code in case of a serious hardware failure. With it, you can track the deployment timeline and revert to a previous version if there are stability problems.

Database

The database should be backed up on a regular basis. Many sites dump the database nightly to a file during low traffic periods. This can be done using an Elgg plugin or using a custom script that runs as a cron job. MySQL has a client program called `mysqldump` that exports the database as a text file:

```
mysqldump -u <user> -p<password> <database> > <dump_file>
```

Once the database is dumped, it can be compressed and transferred to a different location through e-mail, FTP, or rsync. It is best to include the date in the name of the dump files and keep several of the most recent backups around in case an issue is not discovered the same day it occurs.

Files

When a user uploads a profile photo or any other type of file, it is stored in the data directory. If your users upload many videos and photos, then a backup of these files is significantly larger than the back up of the database. One common approach to backing up the files is to use a back-up client on the server to send the files to a tape or backup server. Your hosting provider may provide this service, possibly for an additional cost. There are also commercial companies that provide backup solutions.

If you have a server with multiple disks, then mirroring the files to another disk is sufficient for recovering from most problems. The do-it-yourself approach to file backup is to compress the files and transfer them to another server or set up an incremental backup solution using something like rsync. Be aware that the number of files can grow quickly and transferring an entire backup can take a long time.

Server configuration

You should include any configuration files that you have modified in your backup plan. Often, getting the right configuration is a process of trial and error. Redoing that configuration will not be a pleasant experience, especially after a server failure. Unlike the other backups, this can be done manually whenever the configuration is modified.

Restoring

You do not have a backup system in place until you test the restoration process. This is best performed on a test server. When you test the restoration, be sure to document each step to use as a guide if a failure does occur.

Log rotation

Your server should be set up to automatically rotate the Apache and MySQL logs. Rotating a log file involves backing up the current log file, compressing it, removing older log files, and starting a new log file. This keeps the log files manageable in size and provides a record of recent activity.

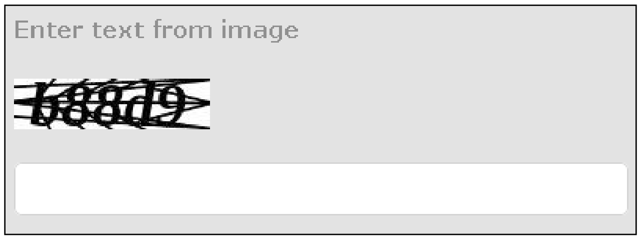
Beyond those log files, there is the Elgg system log that is stored in the database. Elgg is distributed with a plugin that rotates that log by making a copy of the system log table and then emptying the log table. It does not delete older system log table backups so you will need to occasionally clean those up. If you do not need to keep a record of all activity on the site for legal or corporate reasons, then you can drop the old system log tables. (Be very careful about dropping tables. You could easily drop your entire database.) If you do need to keep the logs as records, then you can export them as SQL dump files and then drop them.

Spam

Every website that allows users to post content has to deal with spammers. A site's first defense against spammers is user registration. After they join the site, the second line of defense is detecting and deleting their spam posts, comments, and messages along with banning or deleting their accounts. The best defense is a layered one that uses many different approaches.

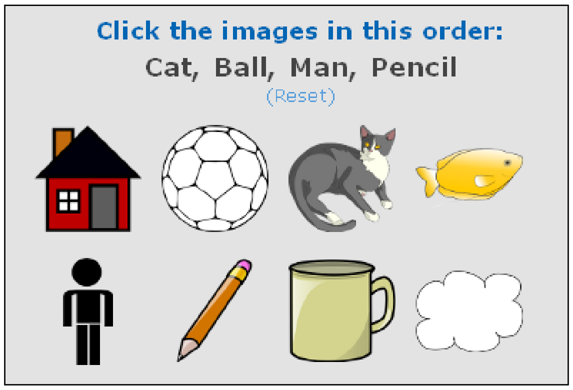
Registration

You want to make it difficult for spammers to automatically generate accounts. This is often done with a **captcha**, which is a test that humans can usually pass but that computers have difficulty with. The most common captcha is a set of letters or words that are distorted or partially obscured as shown in the following image:



This type of common captcha used on many sites has been defeated by spammers. They have been able to write code to automatically fill in the correct text. There are even services available on the web where spammers can submit the captcha images to be solved in exchange for small sums of money.

Another captcha is a visual captcha that asks the viewer to select images based on words:



There are also honeypot captchas that insert a hidden field in the registration form. As a spambot looks at the HTML code, it fills in that field in the form while a human visitor would not. Another captcha technique is asking the user to solve a simple math problem before the registration form is processed.

Even if these anti-spam captchas could prevent every single spambot from creating an account, spammers will still get through the defenses and create accounts. Paid human spammers are a growing trend in the spam business. Incredibly enough, spammers are hiring people to solve the captchas and leave spam on sites.

The last registration spam fighting technique that we will cover is blocking registrations based on IP address. Both spambots and human spammers tend to use the same IP address as they leave spam on many different websites. There are services available that maintain a blacklist of these IP addresses. A popular plugin that uses an IP blacklist is the **Spam Login Filter** (`http://community.elgg.org/pg/plugins/project/774755`). The plugin can submit the IP address to the **Stop Forum Spam** service and also has its own internal blacklist.

As you consider how to limit spammers through registration, be aware that the more difficult that you make it for spammers to register, the more likely it is that you will also turn away real users.

Detecting spammers

After spammers create accounts, they post links on their profiles, create new blog posts, post comments, and send messages to other users. There are three main techniques available for detecting the spammers at work. First, there are plugins that submit the posted content to spam detection services such as **Akismet** (`http://community.elgg.org/pg/plugins/project/723965/`). Second, there are plugins that limit how quickly people can post content as spammers often make tens or hundreds of posts in a short time. Third, your users can report spammers through the reported content plugin that is distributed with Elgg. Once a spammer is reported, a site administrator can ban the spammer or delete their account.

Another common spam fighting technique is adding a `nofollow` tag to links. This tells search engines not to use this link when calculating the rank of the site that the link is pointing to. Spammers are less likely to attack a site when they do not get the benefit of raising their rank in search engines. Elgg adds the `nofollow` tag to all the links submitted by users by default.

Web analytics

While not required, most administrators will want to track the traffic on their site: how many people are visiting the site, where are they coming from, are they returning, and what pages are they viewing? There are two main approaches for web analytics: processing log files and tracking visitors using embedded JavaScript. A popular log-based solution is the open source application AWStats. It periodically processes Apache's access log and compiles several reports about a site's traffic that can be viewed with a web browser. For embedded JavaScript solutions, Google Analytics offers a wealth of information and there are plugins available for integrating it with Elgg. Two self-hosted alternative to Google's offering are the open source packages Piwik and Open Web Analytics. Besides these, there are numerous other free and paid solutions for web analytics.

Daily tasks

With an active user base, there are administrative tasks to be performed each day. The types and amount of administration depend on the user base. Possible tasks include the following:

Monitoring user registration

Occasionally users may have difficulty registering and activating their accounts. This may be because of captchas on the registration form, spam filters catching the validation e-mails, or any number of other problems. Your site should provide a method for a prospective user to contact an administrator to report an issue. You can also check on invalidated accounts with the user validation by e-mail plugin distributed with Elgg.

Reviewing reported content

If you are using the reported content plugin that is distributed with Elgg, then you will want to review the reports on a regular basis. On most sites, members report spam or other users who are disruptive to the community. It is best to maintain a page with the site's policies concerning advertising and community behavior standards to make expectations and consequences clear for the users.

Responding to feedback

Many sites use a plugin to collect feedback from users. The feedback could be reports of bugs, suggestions for improvements, or expressions of appreciation. Responding and interacting with users makes them feel like they are a part of the community.

Community management

Managing the community can take many forms. It can be dealing with troublesome users, updating help pages to answer common questions, reviewing groups for possible duplicates, or involving users in the decision making of the site.

Testing, upgrading, and moving a site

At various times during the life of your site, you will want to test new plugins, upgrade to a new version of Elgg, or move to a different server. This section provides a guide for these activities along with pointers to help you avoid common mistakes.

Testing

It is never a good idea to activate a new plugin on a production site without first testing it. This is one reason why you should have a test site. A test site allows you to try new plugins, test upgrades, perform stress testing, and evaluate performance tweaks without disturbing your users. The test site should mimic the production site as closely as possible including server configuration, code, and data. Depending on available resources, it could be on a server configured exactly like the production server, on the production server, or on any other computer. If you do put your test site on your production server, then do not use it for load testing or trying server configuration modifications.

Mirroring the production site

In order to create a realistic testing environment, you should copy all of the data from your production site to your test site. There are detailed instructions on the Elgg wiki for this (`http://docs.elgg.org/wiki/DuplicateInstallation`), but the major steps are as follows:

1. Perform a clean install of Elgg for the test site.
2. Export the production database.
3. Import the database dump for the test site.
4. Modify parameters in the database that are site specific.
5. Copy users' files from the production site to the test site.

When the test site has the same code and data as the production site, you can be more confident that any testing results will translate to the production site.

Test plan

Adding, changing, or removing code on the production site should be preceded by testing. The best way to systematically evaluate the changes is to develop a test plan that covers all of the functionality that could be affected. In its simplest form, a test plan can be a checklist with a list of user actions and expected responses.

Test plan for blog upgrade

Action	**Expected Response**	**Success**
Save blog post	Create post and forward to view of the post	✓
	Send notifications to user's friends	✓
	Add entry to activity stream	
Save with video	Embedded video displayed left justified	
Save without title	Display error message and forward to input form	

The testing can be manually performed by the maintainers of the site. To accurately replicate user interactions, an account with user privileges should be used for testing rather than an administrator's account. It is also important to test using different browsers to capture any browser-related issues.

There are also tools that automate testing of web applications. You can find these types of tools on the web by searching for *web functional testing* or *web acceptance testing*. By automating the tests, you can develop a suite of tests to run every time a modification is made before rolling the update out to the production site. A popular open source testing tool is Selenium. It has a Firefox add-on for recording tests, which can save a lot of time.

Redirecting e-mail while testing

If your test site uses the mirrored database from the production site, then testing can cause notification e-mails to be sent to your users. You will want to disable e-mail or redirect all messages to a single account. There is a plugin for this: `http://community.elgg.org/pg/plugins/costelloc/read/404328/email-override-for-testing`.

Upgrading

The developers of Elgg release a new version every few months. Sometimes the new releases are focused on fixing bugs and other times they have significant new functionality. Some releases require modifications to existing plugins. If that is the case, it takes time for plugin developers to release versions of their plugins that work with the new release of Elgg. The best policy is to wait until all of your plugins have been updated before you upgrade.

When you are ready to upgrade, perform a dry run of the upgrade on the test site, as follows:

1. Mirror the production site on your test site (code, database, and data directory).
2. Turn off all plugins not distributed with Elgg.
3. Copy the new Elgg code over the old code on the test site.
4. Run the upgrade script: `http://example.org/upgrade.php`.
5. Browse the test site looking for problems. Report and deal with the problems as they are found.
6. Copy over any new plugin code and activate.
7. Repeat step 5 for the plugins that were updated.

If there are any serious issues with the upgrade, then it is best to determine what caused the problem, fix it, and run through the complete upgrade process again on the test site. Once the upgrade goes smoothly (and it usually does), you can work through a test plan looking for bugs in Elgg or in the plugins. If everything is satisfactory after testing, then you are ready to upgrade the production site. The steps for upgrading are as follows:

1. Back up the current code, database, and data directory files.
2. Copy the new Elgg and new plugin code.
3. Run the upgrade script.

Because of the preparation done with the testing site, the upgrade should work perfectly the first time.

Moving a site

If your site is growing, then you may need to move to a more powerful server. Migrating the site follows the same steps as creating a test site: install a fresh copy of Elgg, replace the database with the modified database from the previous site, and copy the users' files in the data directory. There are three common mistakes made when moving a site, as follows:

1. **Not creating a fresh install of Elgg on the new server**. By the time you need to move your site, you have forgotten all about the rewrite module of Apache or the permissions of the data directory. When people try to move everything without doing a new installation, they do not know whether a problem is caused by the server configuration or an incorrectly set parameter in the transferred database. It is better to do the move one step at a time.
2. **Forgetting to update site configuration in the database**. The Elgg database stores the site's base URL and the locations of the code and the data directory. Those are often server specific and if they are not changed, the new site will not work properly. Common symptoms of this include not being able to log in or an oddly formatted site without a theme.
3. **Permissions issues on the data directory**. The web server needs to have read and write permissions on the data directory. With Linux, the web server runs under a special user account that is distribution specific. When moving files from one server to another, it is possible for the file permissions to get confused as the web server user account may be different. The best way to give the web server permissions to the data directory is to change the ownership of the data directory to the web server's user (for example, the user is www-data on Ubuntu).

When moving a site from one server to another, there will be a period of downtime for your users. It is best to practice the move to solve any issues before the actual migration and to develop an estimate of how long this process takes. After the database on the current site is dumped to a file, you should turn off access to the site so that users cannot create new content. There is a plugin available on the Elgg community site for taking a site offline: `http://community.elgg.org/pg/plugins/jdalsem/read/384765/maintenance`. If you are using the same domain name on the new server, then there will be a period of time when DNS servers are pointing to the old site for that domain name. This can be explained to the users before the site migration occurs.

Performance

People evaluating Elgg are often very interested in its performance – sometimes to the exclusion of other considerations. What they do not realize is that growing a website so that it has enough traffic to affect site performance is much more difficult than tuning or upgrading a server. Very few Elgg sites require anything more than a single dedicated server (and many never end up requiring a dedicated server).

Benchmarking

The most important step for increasing performance of your Elgg website is benchmarking. Without recording performance measures, you cannot tell what the current bottleneck is and you cannot predict the bottlenecks of the future. This knowledge is necessary for addressing performance issues.

Just like many of the other topics covered in this chapter, entire books have been written on this topic, so this is a cursory overview. Several resources are listed at the end of the section for those who want to dig deeper into this.

Monitoring and data collection

You do not want to wake up one day to find out that your server is maxed out. By tracking server statistics over time, you can predict when the next performance upgrade should happen based on historical data and determine what the likely bottleneck will be.

Data

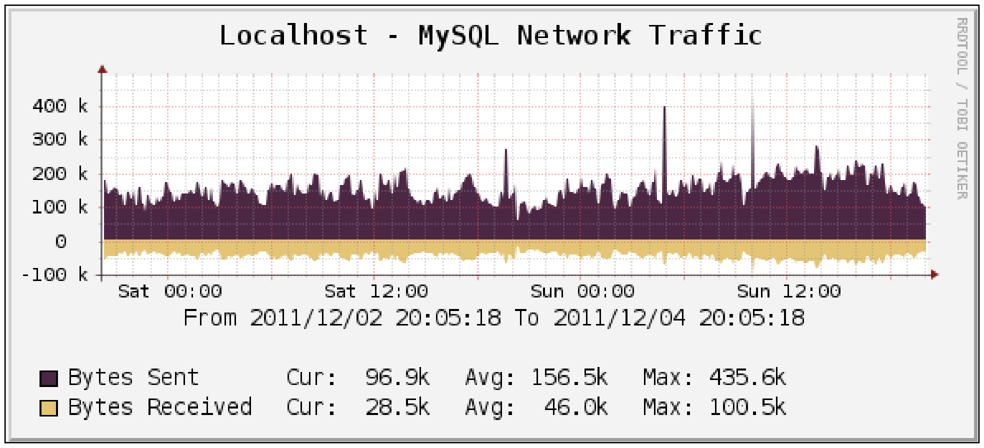
At a minimum, you should be collecting the following:

- Processor usage
- Memory usage
- File I/O
- Network bandwidth utilization
- Database statistics

The data should be collected and logged on a regular schedule. If the data is available in report or graph form, then it will be easy for the administrator of the server to notice trends or problems.

Tools

There are tools available for collecting this data, processing it, graphing it, and sending alerts based on it. **Cacti** (`http://www.cacti.net/`) is an open source tool for graphing server statistics that are collected by RRDtool. The following is an example plot of database activity using Cacti:



Additional performance statistics packages include Munin and ZenOSS. There are also a number of command line tools for collecting performance statistics such as vmstat, iostat, netstat, sar, and mytop.

Stress testing

If you have not gone live with your production server or you have a test server, then stress testing is a great way to evaluate modifications to the server, estimate how much traffic the server can handle, and find bottlenecks. A stress test uses a tool that requests web pages from the server to simulate the load created by many visitors.

Data

These automated stress testing tools are designed for measuring **latency** and **throughput**. Latency is how long a user has to wait to see the requested web page. Throughput is the number of requests that can be served per second. (Throughput can also be measured in the number of bytes per second served.) In addition to collecting the output of the stress testing tool, you should also collect information about what is happening on the server (memory usage, database stats, and so on) to determine where any bottlenecks exist.

Tools

There is a wide range of automated stress testing tools available, ranging from command line tools that repeatedly request a single HTML page to GUI-based tools that can automate tasks such as logging in and posting a blog entry. It is recommended that you run the tool on a different server than the one being tested. This eliminates any competition between the testing tool and the web server processes. If the two servers are close in a network sense that also eliminates any latency issues caused by a slow network connection. Popular tools include the following:

- ApacheBench (`http://httpd.apache.org/docs/2.0/programs/ab.html`)
- httperf (`http://code.google.com/p/httperf/`)
- Siege (`http://www.joedog.org/index/siege-home`)
- Jmeter (`http://jakarta.apache.org/jmeter/`)

Using your web browser is not a good way to benchmark your site. It does not evaluate loading the server, there can be inaccuracies due to browser caching, and network latency can be a problem.

Easy performance gains

Here are three changes that are easy to make that can have dramatic effects on the performance of your site. The first is turning on Elgg's simple cache on the site administration page. The simple cache stores copies of files that Elgg normally would have dynamically created with each request. This saves memory and processing power that can be used to serve other requests.

The second change is installing an opcode cache for PHP. Each time a web page is requested, the Elgg's code is loaded, compiled to opcodes, and then executed. Compiling can consume a significant percentage of the page creation time. An opcode cache stores the compiled code in memory so that the opcodes can be used without passing through the compilation step. This drastically improves performance and is recommended for any production server. There are several opcode caches available for PHP. APC is a popular option and has been used on many large Elgg sites. The default configuration works well so the only setting you may want to tweak in the beginning is `apc.shm_size` based on how much memory your particular Elgg code base requires. The function `apc_sma_info()` can be used to evaluate memory usage.

The third is upgrading your server. If you are using a VPS package, try moving up a tier or two. With a dedicated server package, increase the amount of processing power and RAM. This is the most expensive and involved change so you only want to consider it after the trying the other two first.

Advanced performance tuning and scaling

More advanced tuning of servers or scaling by using multiple servers requires significant technical knowledge. Both Apache and MySQL can be tuned to increase performance for your particular server and your site's traffic patterns. Be aware that a set of parameters may work well under some conditions and cause poor performance under others. Testing configuration parameter changes under different loads to validate the modifications is recommended.

Caching

Caching is one of the most common techniques to improving performance of a web application. Caching is storing data that was dynamically generated so that the next time it is requested, the data is loaded from the cache rather than generated again. Caching is a trade-off. In exchange for using more memory or file I/O, the demands on the server's processors decrease.

Caching occurs throughout the process of requesting, creating, and receiving a web page:

- **Browser caching** lets a browser skip requesting a resource like an image or JavaScript file that it has already downloaded. This is configured by setting directives for Apache. Elgg also sets caching headers on CSS and JavaScript files.
- A **proxy cache** like Squid can be used to quickly serve static files such as images rather than burdening Apache with those types of requests.
- **PHP opcode caching** and **Elgg's simple cache** were discussed in the previous section.
- **Elgg's query cache** stores query results per page load. If the same query is executed more than once when generating a web page, then Elgg uses the results stored in its own memory cache rather than asking for the data from MySQL. This does not help with queries that occur on different web requests because the query cache only exists for a single page at a time.

- **Memcache** can replace Elgg's query cache. The query results are stored in memory and stay resident after any page has been created. This is much more effective than Elgg's native query cache. Memcache can be run on the web server or on its own server. Elgg's Memcache implementation is considered experimental at the time of this writing.
- **MySQL's query cache** also stores the results of queries in memory, but the cache is invalidated if the contents of a table used in the query have changed.

Caching data does not guarantee faster performance. With a small query cache and a large number of unique queries, it is not likely that the needed query is in the cache. This is called a cache miss (as opposed to a cache hit). When this happens, resources are used to check the cache, retrieve the data from the database, and add the result to the cache. The second step happens whether or not caching is used. The other two steps add useless overhead. In this scenario, it would be faster to not use a cache.

Multiple servers

When the resources of a server are exhausted and there is no more tuning or tweaking to do, there are two choices: get a faster server or spread the load over more than one server. Upgrading the server is called vertical scaling while using multiple servers is horizontal scaling. There is a limit to the amount of vertical scaling possible and it becomes increasingly expensive as that limit is approached. Horizontal scaling is cheaper and offers a higher performance ceiling at the cost of adding complexity to the system. The best choice for a particular site depends on the current load and expectations of future growth.

Resources

It is very difficult to write a book on the the topic of performance tuning because the range of required skills is so broad. There are a few general books that can serve as references but they are not written for the beginner. "*Web Performance Tuning*", *Patrick Killelea*, *O'Reilly Media*, is a good, if dated, primer in this area (it was written in the Internet Explorer 5.5 and Netscape 4 era). "*Building Scalable Web Sites: Building, Scaling, and Optimizing the Next Generation of Web Applications*", *Cal Henderson*, *O'Reilly Media*, contains a lot of great information and is intended for a technically proficient audience.

A good web resource for tuning Apache is its documentation (`http://httpd.apache.org/docs/2.2/misc/perf-tuning.html`). Furthermore, most books on the Apache web server include a chapter on performance tuning. MySQL has a wealth of material available starting with "*High Performance MySQL: Optimization, Backups, Replication, and More*", *Baron Schwartz, Peter Zaitsev, Vadim Tkachenko, Jeremy Zawodny D., Arjen Lentz, Derek J. Balling, O'Reilly Media*. Another excellent resource is the MySQL Performance Blog at `http://www.mysqlperformanceblog.com/`.

Reporting bugs

When you or a user comes across a bug, there are a few steps to follow to help developers fix it.

Steps to reproduce the problem

The best bug reports not only describe the problem, but list what steps can be followed to replicate the problem. Here is an example of a bad bug report:

> *The blog plugin doesn't work. It loses my text.*

And here is a good report:

> *All the text I just typed disappeared when I previewed the blog post. This happens regardless of what browser I use and I'm using Elgg 1.7.2.*

Here are the steps to reproduce this:

1. Click on **Write blog post**.
2. Enter text into the body area.
3. Click on **Preview**.
4. Click on **Return to Edit**.
5. The text is all gone.

Not only does the good report list the steps for reproducing the bug, but it also mentions the version of Elgg the problem occurs with and whether this is browser dependent. Bug reports that have this type of information save time because a developer does not have to write back with obvious follow-up questions.

Elgg or plugin?

There are many times when a user of Elgg reports a bug to the Elgg developers that turns out to be caused by a bundled plugin. (A bundled plugin is any plugin not distributed with Elgg. They are also called third-party plugins.) The best way to test for this is to temporarily turn off all bundled plugins and see if the problem still occurs. It is also a good idea to have a test site set up for this. If the bug is still there with the plugins turned off, it could be due to Elgg or it could be due to a server configuration issue. If you are not sure, then the best place to ask for help is the forums on the Elgg community site.

Reporting the bug

If the bug is in Elgg or a plugin distributed with Elgg, then it should be reported to Elgg's bug tracker at: `http://trac.elgg.org/`. The bug tracker requires that you create an account before you can submit a bug report. After it is submitted, you will receive e-mails related to the status of this ticket such as questions from the developers or a notification when it is fixed. Do not expect that the bug will be fixed the same day you report it. Bugs are prioritized by how severe they are and how many people they affect. Depending on the bug, it may be days, weeks or months before it is fixed.

If the bug is in a plugin that you downloaded from the Elgg community site, then you should report the bug to the developer who wrote it. The easiest way to do this is to leave the bug report as a comment on the plugin's web page. This notifies the developer and lets other people see the report if they are having a similar problem.

Summary

Using Elgg in a production environment involves a wide range of skills. There is selection and configuration of the server. The Elgg application and its users require administrative attention. Backups must be made in case anything goes wrong with the server. There is also performance tuning and upgrade planning to handle growth. The amount of time and effort required depends on factors such as the criticality of the site and its growth. If a growing user base is causing you to spend time with the MySQL manual (or hire someone to tune your server), then it is a good problem to have.

A
Developer's Quick Start Guide

This is a short developer's guide for the Elgg framework from a top down perspective. It explores the concepts behind Elgg's design and introduces the design patterns used by Elgg. It also describes the core components that a developer needs to know. For a more tutorial-based introduction to developing with Elgg, see *Chapters 7-9*.

The intended audience of this guide is developers who are building a website with Elgg or who are interested in building plugins. It assumes knowledge of PHP and fluency with software development terminology.

As a quick start guide, it does not cover topics in detail. Instead, the emphasis is on providing a big picture view of Elgg. As each core component is covered, pointers to the source code are given for further investigation. In addition to the documentation in the source code, Elgg's wiki (`http://docs.elgg.org`) is a good resource for learning specifics about how Elgg works.

Overview of Elgg as a framework

This overview covers basic questions about Elgg such as whether it is object-oriented and what kind of template language is used. These are the sorts of questions that are valuable to have answers to before looking at the design and structure of a framework.

What is Elgg?

Elgg is an open source framework for developing social networking and social media websites. It is written in PHP, uses MySQL for data persistence, and includes jQuery for client-side scripting.

Object-oriented or procedural?

The answer is both. The data model of Elgg is primarily object-oriented, while the rest of the framework is mostly procedural. This provides flexibility to plugin developers. For example, a page controller in Elgg can be a script, a function, or a class method. The choice is left up to the developer.

Does it use the Model-View-Controller pattern?

Elgg is not a textbook implementation of the **Model-View-Controller** (**MVC**) pattern. Elgg does not use the terminology of MVC, which can make it difficult at first to see the pattern. Viewing it from an MVC perspective, though, does make it easier to grasp Elgg's design.

Convention or configuration?

The answer to this question is also both. The model and controller use configuration exclusively and the view system is primarily convention-based with some configuration.

Is it extensible?

Elgg has a modular architecture that uses plugins to extend or modify the core engine. Without any plugins enabled, an Elgg site supports account creation, user settings, administration and not much else. This plugin-based approach gives developers flexibility and control when building web applications with Elgg.

Extensibility is also provided through an event-based hook system. Rather than editing core code, developers can modify the behavior of the framework by registering callbacks for an event. The callbacks perform their own processing of the data, prevent the core from taking an action, or change the output of a function. For example, every time a blog post is saved, an event is fired. A callback function registered for that event could check for spam and reject the post.

What template engine is used?

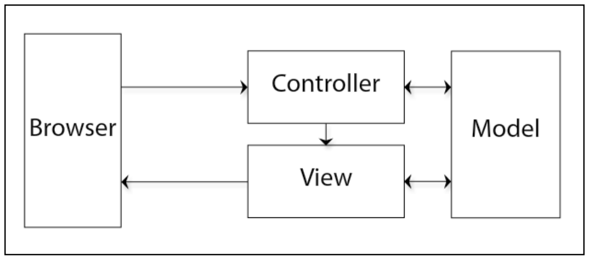
Elgg uses PHP as its template engine. This results in a flexible view system since the full power of PHP is available. Developers also do not have to learn a new template language to use Elgg as they would with an engine like Smarty. On the downside, an expressive template language such as PHP is a temptation to mix controller code into the views.

A Model-View-Controller perspective of Elgg

This section provides an overview of Elgg through the lens of the MVC pattern.

Overview

It all starts with a request. The most common scenario is a web browser requesting an HTML page. Let's assume it is a request for the web page that displays Joe's latest vacation photos. The request arrives at the controller. The controller confirms that Joe exists as a user at the site. It tells the model to increment the view counter on the photo album. The controller then passes the album identifier over to the view for rendering. The view pulls the photo objects from the model and creates the HTML page that is sent to the web browser.

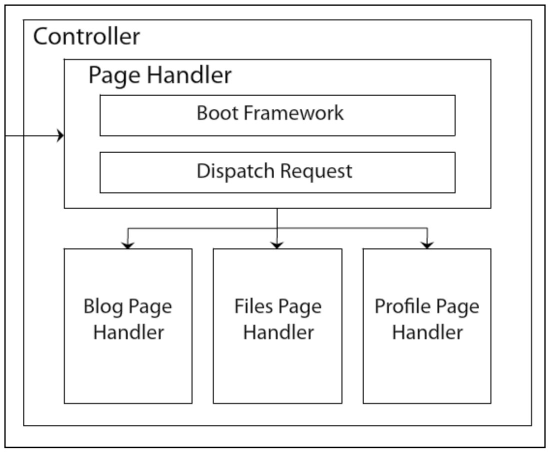


Controllers

Elgg uses the **Front Controller** pattern with a twist. In the Front Controller pattern, every request is routed through a single controller. This is unlike web applications that use individual scripts to handle different types of requests (for example, logging in uses `login.php`, posting a comment uses `post-comment.php`, and so on). The twist with Elgg's implementation of this pattern is that it has two primary front controllers. One front controller handles actions, which usually involve the submission of a form. The other front controller handles requests for pages, which are most often HTML pages.

The job of the front controller is loading the framework and dispatching the request to the appropriate secondary controller. A page controller processes the parameters of the request, updates the data model, and turns the processed request over to the views system. An action controller is similar except that it does not turn a request over to the views system, but instead forwards the requester to a new page. In Elgg, these controllers are called **handlers**.

The following diagram shows request processing logic inside the controller box of the overview diagram. The diagram includes three of the page handlers provided by plugins as examples of secondary controllers.



The two primary handlers are located at `/engine/handlers/`.

Model

The model of Elgg consists of three parts:

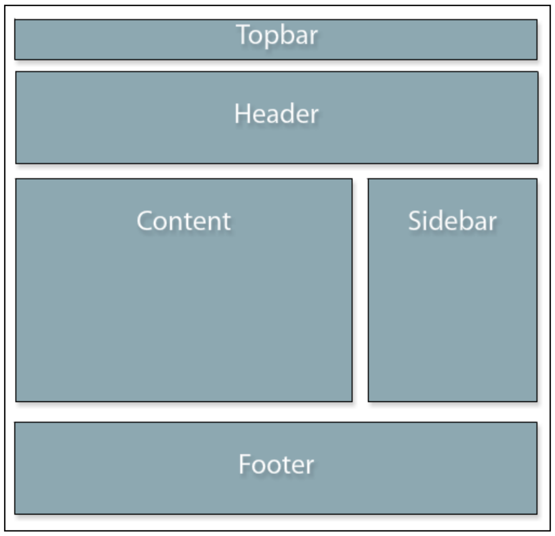
1. Data model classes that manage entities, metadata, and relationships.
2. Helper functions that build queries based on parameters (*give me the 10 latest blog posts written by Bob*).
3. Database access layer that provides an interface to the MySQL database.

The data model supports a wide range of use cases because of its simple and flexible design. Developers should use Elgg's data model rather than writing their own queries or creating their own tables.

Views

The views system renders data into HTML (or other formats such as RSS). It has two steps. In the first step, the controller requests the view system to render the data from the model for presentation. In the second step, this output is inserted into a layout and the complete response is rendered. This is different from frameworks that use a per page template that lays out an entire page. An advantage of the two step approach is the ease of maintaining a consistent look across all pages.

The output from the first step is represented by the content box in the following diagram. The second step handles the remaining sections of a page.



Both steps use templates which are called **views** in Elgg. Views are short scripts that provide building blocks for creating web pages and other types of output. In the preceding diagram, the topbar, header, sidebar, and footer are created by views. As mentioned earlier, PHP is the template language used in the views.

Each request has a view type that determines the response format. If the view type is *default*, then an HTML page is created. If it is *json*, then the same data is rendered in JSON format. The view type is set by the controller based on the parameters of the request. This view type functionality can be used to serve a mobile-specific layout to mobile devices. An example of this can be found in *Chapter 9*.

Routing

The first stage of routing happens in Apache. Elgg uses rewrite rules to map a request to one of the primary handlers. The primary handler dispatches the request to the registered secondary handler.

As an example, the relative URL `/action/comments/add/` is routed to the action handler registered for the `"comments/add"` action. The relative URL `/friends/johndoe/` is routed to the page handler registered for the `friends` identifier.

Code location

Apache rewrite rules: `/.htaccess`

Primary handlers: `/engine/handlers/`

Library functions: `/engine/lib/actions.php`, `/engine/lib/pagehandler.php`

Actions

Actions are generally requests to update the data model. Examples include posting a comment, deleting a blog post, or joining a group.

Action handlers are registered by mapping a portion of a URL to a script:

```
elgg_register_action('blog/save', "$action_path/save.php");
```

The `save.php` script is included (executed) by the function `action()` when a request is made against the `/action/blog/save/` URL.

The normal flow of an action handler is as follows:

1. Access user-submitted data using `get_input()`.
2. Validate data.
3. Update data model.
4. Queue status message using `system_message()` or `register_error()`.
5. Forward the requester to a page handler using `forward()`.

Code location

Core action handlers: `/actions/`

Plugin action handlers: `/mod/<plugin name>/actions/`

Page handlers

Page handlers manage the response to a request. The response can be a web page, RSS feed, or any number of other output formats. Because Elgg uses the MVC pattern, the same handler is used for different response formats; only the views change.

A page handler is registered by mapping a URL identifier to a function, as follows:

```
elgg_register_page_handler('blog', 'blog_page_handler');
```

This handler function is called when a request is made to a URL starting with `/blog/`.

The handler function receives an array of the segments of the URL. This array is used to further route the request in the handler. The handler function can process requests in the function itself, include page handling scripts, or use a page handler class. Developers can select the approach that best matches the job and their programming style.

Page handlers include both controller and view code and work like the following:

1. Access any user input through `get_input()`.
2. Pull data from the model related to the request.
3. Push the data into the views system for rendering.
4. Send output to the requester using `elgg_view_page()`.

Code location

Core page handlers: directories in `/pages/`

Example of script-based page handling in a plugin: bookmarks plugin

Example of function-based page handling in a plugin: blog plugin

Framework booting

At the top of the primary handlers, there is the following line:

```
require_once(dirname(dirname(__FILE__)) . "/start.php");
```

This loads the Elgg framework, which includes the following:

- Loading the core libraries
- Connecting to the database

- Loading Elgg's configuration
- Loading plugins
- Loading language files
- Initializing the user's session
- Registering handlers and callbacks

When the boot process is completed, control is returned to the handler. Elgg's routing and page handling system handle booting the framework before a plugin is called.

Code location

Boot script: `/engine/start.php`

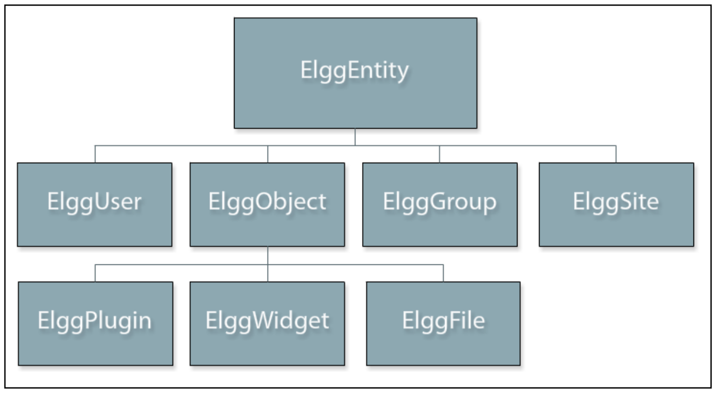
Data model

Elgg has a simple, but flexible data model. It supports entities, relationships, and extenders:

- Entities are roughly nouns. A user, group, or blog post are all examples of entities.
- Relationships connect two entities. Two users are friends. A user is a member of a group. A user is notified when another user posts a comment.
- Extenders describe entities. There are two types of extenders: metadata and annotations. A photo has 10 views. A file has been downloaded 300 times. A user's location is Brazil. These are examples of data that is stored as metadata or annotations on an entity.

Entities

`ElggEntity` is the parent class of all entities. It is an abstract class and is extended by four classes: `ElggGroup`, `ElggObject`, `ElggSite`, and `ElggUser`. The Elgg core also provides three subclasses of `ElggObject` as shown in the entity inheritance chart. Plugins can create classes that extend any of these classes.



Type and subtype

Each entity has a type that corresponds to its primary class. The types are group, object, site, and user. Each entity can have a subtype. The subtypes allow developers to subclass one of the primary classes. For example, an `ElggPlugin` entity has the subtype `'plugin'`. The type and subtype attributes enable the framework to instantiate the correct class when loading an entity from the database.

GUID

Every entity has a unique identifier. It is global for that Elgg instance.

Owner

Entities are owned by other entities. A blog post is owned by the user who wrote it. Widgets are owned by the user whose profile they are on.

Container

Entities can also belong to a container. When a user uploads a file to a group, the file is owned by the user, but its container is the group.

Access

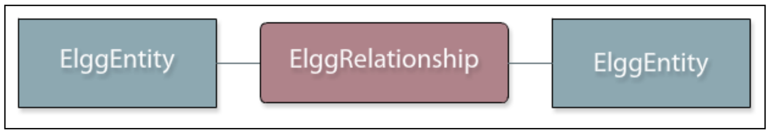
Every entity has an access level. This controls who can view the entity. The access levels provided by Elgg are private (only that user), the user's friends, anyone logged in, and public (everyone).

Database

There are several ways to represent a class inheritance structure in a relational database. Elgg uses the **Class Table Inheritance** pattern. In this pattern, each class has its own table. Elgg has tables for the first two layers of classes: the superclass, `ElggEntity`, and its subclasses. Therefore, loading an entity from the database consists of loading data from two tables. For example, loading an `ElggUser` object loads data from the entity table and the user table. The values from these two tables are called the entity's attributes. The framework does not create tables for developer-created subclasses.

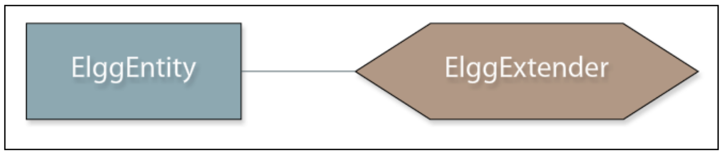
Relationships

The `ElggRelationship` class describes a connection between two entities. It supports arbitrary relationships, as shown in the following screenshot. The first entity is the subject of the relationship and the second is the object. If Bob "friends" Larry, then a relationship is created with Bob as the subject and Larry as the object.



Extenders

The `ElggExtender` class is abstract and is implemented by two subclasses: `ElggAnnotation` and `ElggMetadata`.



Annotations and metadata are both used to extend and describe entities. They have a name and a value. A user with a favorite color of blue could be represented as metadata with the name of "`favorite_color`" and the value of "`blue`". A user downloading a file can be captured as an annotation where the name is "`download`" and the value is `1`.

Annotations tend to describe something about another user's content: ratings, downloads, views, likes. Metadata is used more often to capture information by the owner of an entity: whether comments are allowed on a blog post, what the tags are for a file, or how many friends to display in a profile widget. In addition, annotations support mathematical operations such as calculating the average rating or the total number of downloads.

It is very easy to add metadata as Elgg uses the magic methods `__get()` and `__set()` on the entity class. Using the favorite color example again, the favorite color is set with this code:

```
$user->favorite_color = "blue";
```

Elgg handles the persisting of the metadata in the database.

Entity attributes are different from metadata

Each entity type has a set of attributes that are persisted in the entity database tables. Type, subtype, create time, and access level are all attributes. Attributes and metadata are accessed through the same set and get methods:

```
$access_level = $file->access_id;
```

They act differently though. When an attribute is changed, the entity's `save` method must be called to persist the change. In addition, some attributes cannot be changed (guid, type, and subtype). One last hint with attributes is to always use the convenience method `getSubtype()` as it returns the string representation of the subtype rather than accessing it directly (`$entity->subtype`).

Database

Elgg uses the **Entity Attribute Value** model for storing metadata and annotations. This model supports the dynamic addition of metadata as demonstrated in the favorite color example.

The annotation and metadata tables are normalized. Rather than storing the names and values as strings, they are stored as keys into a strings table.

Retrieval functions

Elgg has a set of `elgg_get_entities*` functions for retrieving entities based on parameters. These functions take a single argument of an array with key value pairs. For example, getting the five latest blog posts on the site is done with a call to `elgg_get_entities()`:

```
$options = array(
  'type'    => 'object',
  'subtype' => 'blog',
  'limit'   => 5,
);
elgg_get_entities($options);
```

The functions support querying by any of the properties that have been mentioned:

- `elgg_get_entities_from_metadata()`
- `elgg_get_entities_from_annotations()`
- `elgg_get_entities_from_relationship()`

In addition, there are parallel functions that both retrieve and display lists of entities. They take the same parameters and begin with `elgg_list_entities`.

Code location

The code for the data model is located in `/engine/lib/` and `/engine/classes/`. The class files all start with Elgg: `ElggEntity.php`, `ElggObject.php`, `ElggUser.php`, `ElggGroup.php`, `ElggMetadata.php`, `ElggAnnotation.php`, and `ElggRelationship.php`.

Entities: `entities.php`, `groups.php`, `objects.php`, `sites.php`, `users.php`, `filestore.php`, `plugins.php`, and `widgets.php`

Relationships: `relationships.php`

Extenders: `extender.php`, `annotations.php`, `metadata.php`, `metastrings.php`

Database access: `database.php`

Views

Elgg's view system creates the response sent to the requester. It could be an HTML page, RSS feed, XML document or any other format. The generation of the response is divided into two parts: rendering the data related to the request and including the rendered data into the full response. Elgg uses a template system to create the response.

View templates

Templates allow dynamic content to be included in a static structure. Consider the following simple example of an Elgg view:

```
<h2><?php echo $vars['title']; ?></h2>
```

This view creates the HTML for displaying a title. The `h2` tag is static while the value between the tags is dynamic.

The `$vars[]` associative array contains the variables passed into the view. The function `elgg_view()` is used to generate the output of a view (a string):

```
echo elgg_view('page/elements/title', array('title' => 'Hi'));
```

The name of a view is determined by its path. A core view with the name 'input/button' is located at `/views/default/input/button.php`.

Views can be nested. The `'page/elements/header`' view includes other views:

```
<?php
/**
 * Elgg page header
 * The header lives between the topbar and main content area.
 */

// link back to main site.
echo elgg_view('page/elements/header_logo', $vars);

// drop-down login
echo elgg_view('core/account/login_dropdown');

// insert site-wide navigation
echo elgg_view_menu('site');
```

Views are not limited to the variables in the `$vars[]` array. They can pull information from the data model or helper functions.

Page shells and layout

Rendering the response is divided into two steps. The first is rendering the data specific to a request and the second step is laying out the page. In code, it looks like the following:

```
// render the latest bookmarks as a list
$options = array(
  'type' => 'object',
  'subtype' => 'bookmarks'
);
$content = elgg_list_entities($options);

// render the response
$title = elgg_echo('bookmarks:latest');
$body = elgg_view_layout('one_sidebar',
  array('content' => $content));
echo elgg_view_page($title, $body);
```

The function `elgg_view_layout()` lays out the central portion of the page identified as `'layout'` in the preceding layout diagram. The output of the layout is passed to `elgg_view_page()` and is inserted into the final response.

Elgg includes generic layouts for one, two, and three columns in addition to a layout specifically for content pages (viewing blogs, bookmarks, files, and similar types of content). The views for these layouts are located in `/views/default/page/layouts/`. Additional layouts can be added through plugins.

The `elgg_view_page()` function uses a page shell view to create the complete response. The default page shell for HTML is located at `/views/default/page/default.php`. This view includes the HTML head, topbar, header, and footer.

View type

The view type parameter determines the response format. A request for `http://example.org/blog/owner/johndoe` returns HTML formatted for a web browser. A request for `http://example.org/blog/owner/johndoe?view=rss` returns an RSS feed. The standard HTML view type uses the default view, which is why those views are located in `/view/default/`. Likewise, the RSS views are in `/views/rss/`. Each top level directory in `/views/` is a different view type.

Overriding and extending views

Overriding a view replaces a core view with a plugin's view. When a view exists in the same relative location in a plugin as it does in the core views, Elgg uses the plugin's view. A plugin that has a file at `/mod/<plugin name>/views/default/page/elements/header.php` overrides the '`page/elements/header`' view of the core. In addition to overriding core views, plugins can override views from other plugins.

The content from one view can be added to that of another view by extending it. The syntax for this is as follows:

```
elgg_extend_view('page/elements/header', 'mytheme/site_menu');
```

Special views

There are certain views that Elgg automatically checks for and uses, if they exist. This is one instance of convention over configuration in Elgg. A primary example of this is that each entity has a special view used to display it. The name of the view is based on the entity's type and subtype. The view for displaying a blog post is `'object/blog'` and the view for a user is `'user/default'`. Rendering an entity is performed by passing it to the `elgg_view_entity()` function, as follows:

```
$html = elgg_view_entity($blog);
```

Other special views include the following:

- Plugin settings: `/mod/<plugin name>/views/default/plugins/<plugin name>/settings.php`
- User settings: `/mod/<plugin name>/views/default/plugins/<plugin name>/usersettings.php`
- Widget settings: `Widget settings: /mod/<plugin name>/views/default/widgets/<widget name>/edit.php`
- Widget content: `/mod/<plugin name>/views/default/widgets/<widget name>/content.php`

Code location

Functions: `/engine/lib/views.php`

Core views: `/views/`

Events and hooks

Elgg's events and plugin hooks are an important part of the platform's extensibility. Plugins register functions that are called when an event occurs or a hook is triggered. Through these callback functions, plugins change or extend the functionality of Elgg's engine.

There is overlap between the functionality of events and plugin hooks. Broadly, they can be described as follows:

- Elgg events notify the registered callback function that something has happened
- Plugin hooks give callback functions the opportunity to change something that the core has done

The callback functions are called handlers, but do not confuse them with page handlers.

Elgg events

Event handlers are registered with the function `elgg_register_event_handler()`:

```
elgg_register_event_handler('pagesetup', 'system',
  'blog_page_setup');
```

Multiple handlers can be registered for an event. Each event is identified by two words (`'pagesetup'` and `'system'` in this example). The handler is passed a single object involved in the event. If a blog post has been saved, then it is the blog object. If a comment has been posted, then it is the annotation object. Returning false prevents any other event handlers from running and may stop the core from carrying out its current task.

Plugin hooks

Plugin hook handlers are registered with the function `elgg_register_plugin_hook_handler()`:

```
elgg_register_plugin_hook_handler('public_pages', 'walled_garden',
  'twitter_api_public_pages');
```

Plugin hook handlers are passed a return value and other data associated with the hook. The handler function can return the original return value, a modified version, or something completely different.

Event handlers and plugin hooks do not all operate in the same manner. It is best to check the code and comments where the event or hook is triggered before attempting to use one. Possible events and hooks are found by searching the code for `elgg_trigger_event()` and `elgg_trigger_plugin_hook()`.

Code location

Functions: `/engine/lib/elgglib.php`

Plugins

The Elgg core does not do much by itself. It provides the foundation and the plugins determine what the web application truly does. Elgg is distributed with more than 30 plugins that are referred to as bundled plugins. Additional plugins are available from the Elgg community site (`http://community.elgg.org/`) and Elgg's Github site (`https://github.com/Elgg`).

Plugins can add significant user-facing functionality or customize minor backend processing. They can override core functionality or extend it. Building a website with Elgg requires writing and using plugins. *Chapters 7* and *8* provide a tutorial-based approach to learning to write plugins.

Plugins are installed in the `/mod/` directory. Each directory in `/mod/` is a plugin. Plugins are structured like mini versions of the Elgg core (though only the classes, languages, and views directories are required to have those names):

- `/actions`
- `/classes`
- `/graphics`
- `/languages`
- `/pages`
- `/vendors`
- `/views`
- `start.php`
- `manifest.xml`

The manifest file describes the function of the plugin and this information is used on the plugin administration page. The `start.php` file is the boot script of the plugin. It registers callbacks, extends views, and performs other initialization.

Initialization

When Elgg loads, it includes the `start.php` file of each activated plugin. In `start.php`, a plugin registers a handler for the `'system'`, `'init'` event:

```
elgg_register_event_handler('init', 'system', 'blog_init');
```

This event is triggered when all of the core libraries and plugins have been loaded. It is in this event handler that the plugin does its initialization:

```
function blog_init() {

  elgg_extend_view('css', 'blog/css');

  elgg_register_page_handler('blog', 'blog_page_handler');

  // Register for search.
  elgg_register_entity_type('object', 'blog');

  $action_path = dirname(__FILE__) . '/actions/blog';
  elgg_register_action('blog/save', "$action_path/save.php");
}
```

This occurs before a request is dispatched to a handler (action or page handler).

Plugin order

Plugins can override views, actions, and language strings. If more than one plugin overrides the same view, then it is the plugin that loads last that has its view used. The same is true for anything else that can be overridden. The plugin order is set on the plugin administration page.

Conventions

An important convention when writing plugins is namespacing the code to prevent conflicts with other plugins or future additions to the core framework.

Functions normally include the name of the plugin:

```
function blog_url_handler($entity) {
```

The same is true for language strings (described later in this guide):

```
'blog:revisions' => 'Revisions',
```

Views should be namespaced by creating a directory under the default directory named after the plugin:

```
/mod/blog/views/default/
                        /blog/
                              /sidebar/
                                       /css.php
                                               /group_module.php
```

Actions should also begin with the name of the plugin:

```
elgg_register_action('blog/save', "$action_path/save.php");
```

In addition to these naming conventions, there are Elgg coding standards included in the docs directory. These are the standards that the core developers follow.

Themes

A theme is a plugin. A theme plugin overrides parts or the entire default theme that is built into Elgg. All themes override elements of Elgg's CSS and some themes additionally override views used to create a site's HTML. A theme may also include graphics or JavaScript code.

Elgg has its own CSS framework that uses the view system. The primary CSS view is `'css/elgg'` and it includes the various modules for layouts, navigation, forms, and other style categories. *Chapter 9* describes the CSS framework and provides a tutorial for creating a theme.

Code location

CSS views: `/views/default/css/`

Activity stream

The activity stream is called the river in Elgg. Adding an item to the river is accomplished through a call to `add_to_river()`:

```
add_to_river('river/object/file/create', 'create',
  elgg_get_logged_in_user_guid(), $file->guid);
```

The first parameter is the name of the view used to display the river item. The view must exist before an item can be added.

Code location

Functions: `/engine/lib/river.php`

Example action: `/mod/pages/actions/pages/edit.php`

Example view: `/mod/pages/views/default/river/object/page/create.php`

Notifications

Users can register for notifications on activity by their friends or in their groups. The core provides e-mail-based notifications and the **messages** plugin adds site-based private messaging notifications. Additional types of notifications can be added through plugins.

A plugin registers with Elgg to have notifications sent whenever one of its objects is created by a user. The registration is performed in the plugin's initialization function, as follows:

```
register_notification_object('object', 'blog');
```

A plugin can set a custom notification message by registering for the `'notify:entity:message','object'` plugin hook.

Code location

Functions: `/engine/lib/notification.php` and the notifications plugin

Internationalization and localization

Elgg uses the function `elgg_echo()` to handle the internationalization of user interface text. The function is passed a string identifier and it is mapped to the appropriate string in the viewer's language:

```
$title = elgg_echo('file:yours');
```

The mappings are stored in language files named according to the language code (`en.php` has the English mapping). The mappings are associative arrays:

```
$english = array(
  'item:site'        => 'Sites',
  'login'            => 'Log in',
  …
  'tags:site_cloud' => 'Site Tag Cloud',
);
```

The language files for a plugin are located in its languages directory: `/mod/<plugin name>/languages/`. A mapping defined in the core or in a plugin can be overridden in a plugin. Plugin loading order determines which string is used if more than one plugin defines a mapping. Plugin language files are automatically loaded by the Elgg engine.

Code location

Core language file: `/languages/en.php`

Plugin language files: `/mod/<plugin name>/languages/en.php`

Lightning round

There are many topics in Elgg development that have not yet been covered in this guide. This section includes brief descriptions along with pointers of where to look in the code for more information.

Authentication

Elgg uses a simple version of **pluggable authentication modules** (**PAM**). The default authentication module uses the username and password available from the `ElggUser` class and stored in the database. Additional authentication modules can be registered through plugins.

See: `/engine/lib/pam.php` and `pam_auth_userpass()` in `/engine/lib/sessions.php`

Caching

There are several types of caches in Elgg. There are memory caches for database queries and loaded objects to reduce the number of database queries. Views can be cached to files to skip the generation of frequently used views like the CSS view. There is also experimental support for memcache. The caching code is spread throughout the engine libraries.

See: `/engine/lib/cache.php`, `/engine/lib/memcache.php`, and any of the data model files.

Configuration

The database username, password, and hostname are stored in `/engine/settings.php`. Other configuration settings are stored in the database. Elgg supports system-wide settings through its data list functions and site-specific settings through its configuration functions.

See: `/engine/lib/configuration.php` and the `config` database table for site settings and the `datalists` database table for installation settings.

Debugging and logging

Elgg provides its own logging function: `elgg_log()`. This function works in concert with the debug mode in the site settings. The debug mode parameter sets the trace level to control the amount of information logged. Elgg supports logging to PHP's error log or to the screen. Additional destinations can be set through a plugin hook.

User actions are logged to the database through the system log functions. There are two plugins for working with the log: **logbrowser** and **logrotate**.

Elgg also overrides PHP's default logging and exception handling with functions in `elgglib.php`.

See: `/engine/lib/elgglib.php` and `/engine/lib/system_log.php`.

JavaScript

Elgg includes its own library for client-side JavaScript and Ajax functionality built on top of jQuery. It is designed to be extensible with plugins able to create their own namespaced objects (see `/mod/embed/views/default/js/embed/embed.php` for an example of that). The library supports submitting to Elgg actions via Ajax, displaying status messages, and custom client-side events.

See: `/js/ and /views/default/js/`

Menus

Elgg has many menus. Site-wide navigation, avatar drop-down menus, and a footer menu are just a few examples. All of the menus are created using a single API. This API supports static menus, context-specific menus, custom templates, and hierarchical menus. A valuable resource for understanding the menu system is a series of articles posted on the Elgg blog. They can be found by visiting `http://blog.elgg.org` and searching for "`menu`".

See: `/engine/lib/navigation.php` and `/views/default/navigation/menu/`

Private settings

Private settings are similar to metadata and are used for storing settings for plugins and users.

See: `/engine/lib/private_settings.php`

Search

Search is provided through a plugin that uses MySQL's free text search capabilities. There is a readme file in the plugin's directory that provides an overview of the plugin and how to extend it.

See: `/mod/search/`

Security

A wide range of topics fits under the heading of security. This section highlights Elgg's security against cross-site scripting (XSS), cross-site request forgeries (CSRF), and SQL injection attacks. User-submitted input passes through the `get_input()` function, which filters the data. The filtering occurs through a plugin hook that the **htmlawed** plugin handles. The action system uses a token-based approach to CSRF attacks. SQL injection protection is provided by escaping parameters during query generation. More detailed information is available on the Elgg wiki.

Session handling

Elgg uses PHP's session handling and stores the session data in the database. A session contains the user object for the logged in user, which is accessed through the function `elgg_get_logged_in_user_entity()`.

See: `/engine/lib/sessions.php`

Unit tests

Elgg uses the `SimpleTest` framework for its unit tests. The unit tests are run through the **diagnostics** plugin. Plugins can add unit tests by registering a callback for the `'unit_test', 'system'` plugin hook.

See: `/engine/tests/` and `/vendors/simpletest/`

Web services

A REST/RPC hybrid web services API is included with Elgg. It enables sites to expose a custom web services API. These web services can be used for building desktop and mobile applications, integrating with third party applications, or creating mashups with other websites.

See: `/engine/lib/api.php`

Widgets

Elgg has a simple widget framework. By default, widgets are available on users' profiles and dashboards. They are easy to create and there are tutorials in *Chapter 8* and on the Elgg wiki for building them.

See: `/engine/lib/widgets.php`

Summary

Every framework has its own terminology. Learning that terminology makes it easier to read its documentation and communicate with other developers. The following is a list of common development terms for Elgg:

- **Action**: A controller that primarily handles form submissions by updating the data model.
- **Core**: All the code distributed with Elgg. Core views refer to views located in `/views/`. Core actions are those in `/actions/`.
- **Engine**: The libraries and classes in `/engine/`.
- **Event**: A change in state that results in registered handler functions being called.
- **Handler**: A function that is called when an event or hook is triggered. Also, a controller.
- **Plugin hook**: An event-driven mechanism for adding to or modifying the core engine.
- **River**: The activity stream.
- **Third-party plugins**: Plugins not distributed with Elgg.
- **View**: Templates file in the views system.

B
Views Catalog

The catalog is intended for two audiences:

1. **Themers**. The catalog provides a visual overview of the view components available for styling. The location of each view is listed so that the developer can examine its HTML structure. In addition to adding CSS to style the default HTML of the view, themes can override views to change the HTML. Advice is included with the views for creating themes.
2. **Plugin developers**. Building plugins almost always involves using the views system. The catalog is a quick reference guide to the available views. Many of them include implementation details that can save a developer time when learning Elgg.

The views are divided in this appendix into functional groups such as layout, navigation, forms, and widgets. Each view section has a short description with a screen capture of the view in action. This is followed by the location of the view and tips about using it or styling it.

This appendix can be used in combination with the theming sandbox included in the Elgg developer tools plugin. The sandbox provides themers the opportunity to experiment with styling elements of Elgg's view system in a controlled environment.

Using views

Before using a view for the first time, check the documentation at the top of the view file. The documentation lists its parameters and additional information about the view. For example, here are the comments from the "`input/plaintext`" view:

```
/**
 * Elgg long text input (plaintext)
 * Displays a long text input field that should not be overridden
  by wysiwyg editors.
 *
 * @uses $vars['value']      The current value, if any
 * @uses $vars['name']       The name of the input field
 * @uses $vars['class']      Additional CSS class
 * @uses $vars['disabled']   Input should be disabled
 */
```

Views listed in the catalog are considered core views and are found in `views/default/`. The only exception being the search views that are in `mod/search/views/default/`. As a reminder, the name of the view is related to its filename. The `input/plaintext` view is equivalent to the file `views/default/input/plaintext.php` (as long as it is not overridden).

Page structure

By using the same views on every page, Elgg makes it simple to maintain a consistent look and feel across the site. Creating a new theme often starts by styling or overriding these **page** views. They are organized into four divisions: page shells, layouts, elements, and components. They are all found in the `/views/default/page/` directory.

Shells

A page shell defines the overall structure of a web page. The topbar, header, and footer are all included by the shell. Elgg is distributed with two primary shells. The default shell is used on user-facing pages and the admin shell is used in the administrative backend of Elgg.

Views: `page/default`, `page/admin`

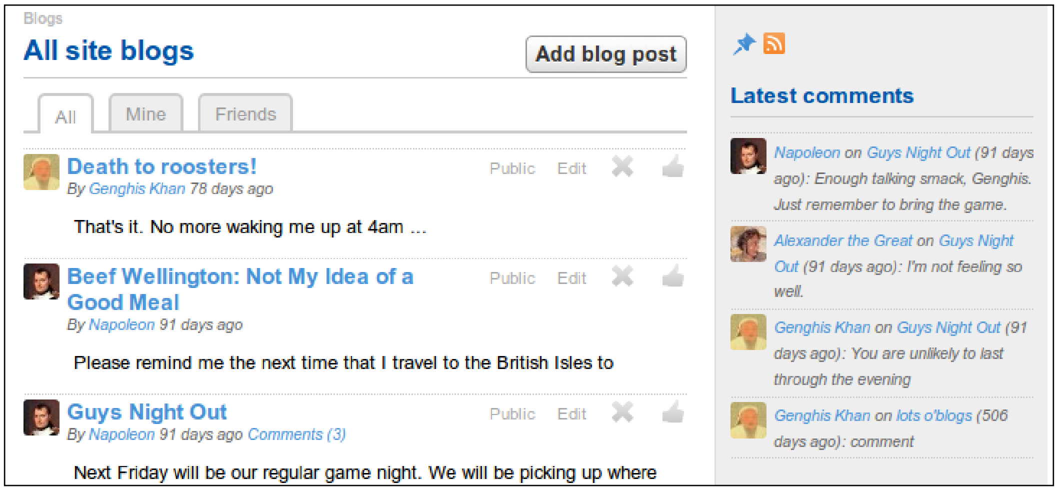
Developers: If you wanted a few pages to have a very different layout that the rest of the site, then a good way to accomplish this is by creating a new page shell. The page shell is selected when calling `elgg_view_page()`.

Themers: If you need to override the default page shell, then remember to include the "`page/elements/foot`" view as it is used by Elgg's JavaScript library for loading JavaScript files.

Layouts

While the page shell defines the areas that remain fairly constant across the site, the layout contains the dynamic content. Elgg has three structural layouts: one column, one sidebar, and two sidebars. The most common of these is the one sidebar layout with a main content area and a single sidebar that is on the right in the default theme.

In addition to the structural layouts, Elgg is distributed with admin, content, and widgets layouts. The content layout is used by all the major content plugins such as blogs and file. It adds additional structure to the one sidebar layout with action buttons and tabs (the filter menu). The following is an example of the content layout on a bookmarks page:



Death to roosters!
By Genghis Khan 78 days ago

That's it. No more waking me up at 4am ...

Beef Wellington: Not My Idea of a Good Meal
By Napoleon 91 days ago

Please remind me the next time that I travel to the British Isles to

Guys Night Out
By Napoleon 91 days ago Comments (3)

Next Friday will be our regular game night. We will be picking up where

Napoleon on Guys Night Out (91 days ago): Enough talking smack, Genghis. Just remember to bring the game.

Alexander the Great on Guys Night Out (91 days ago): I'm not feeling so well.

Genghis Khan on Guys Night Out (91 days ago): You are unlikely to last through the evening

Genghis Khan on lots o'blogs (506 days ago): comment

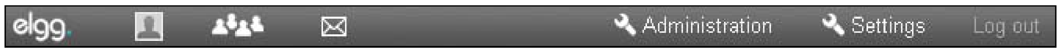
Views: `page/layouts/*`

Elements

Page elements are views that are typically used a single time when creating a page. Page shells and layouts use them to define the structure of a page. All of the views included in this section are in the `views/default/page/elements` directory except for the search box, which is in the search plugin.

Topbar

The topbar is the tool bar at the top of every screen. The links in the topbar are managed through the topbar menu, which is discussed later in the appendix.

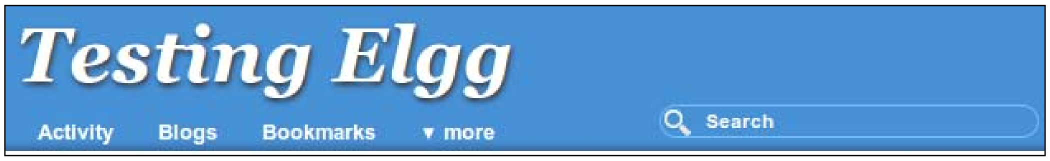


View: `page/elements/topbar`

Themers: When overriding this view in your theme, consider whether you want to continue using the topbar menu, site menu, combine the two menus, or create a custom set of links. The decision that you make affects the integration of plugins. If a new plugin expects the topbar menu to exist and you have replaced it with a hard-coded menu, then you will need to manually update that menu.

Header

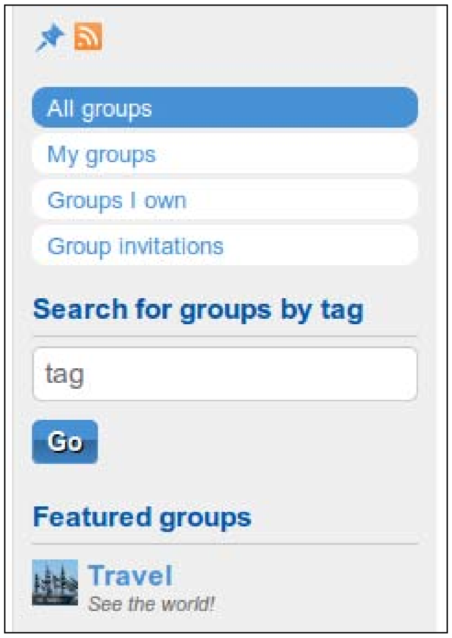
The header includes the site title, site navigation, and a search box. The search box can be removed with a call to `elgg_unextend_view()`. Content can be added to the header by extending its view or overriding it.



View: `page/elements/header`

Sidebar

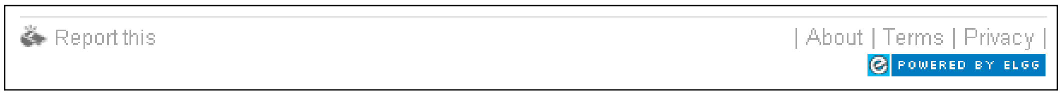
The sidebar view contains the owner block and the page menu. It can be extended to add additional content.



View: `page/elements/sidebar`

Footer

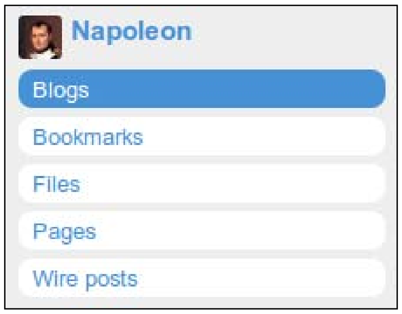
Many themes override this view to create a custom footer. Links are added to the footer through the footer menu described later.



View: `page/elements/footer`

Owner block

This view includes a profile photo of a person or group along with content links. It appears at the top of the sidebar in the default theme on some content pages.

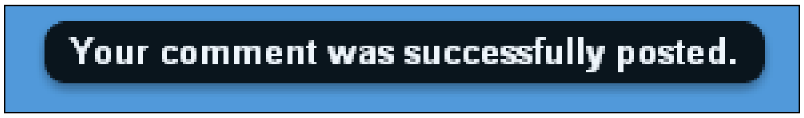


View: `page/elements/owner_block`

Developers: Links are added to the owner block by using the `'register'`, `'menu:owner_block'` plugin hook.

Status messages

When a user performs an action (logging in, posting a comment), the next web page includes a status message at the top of the page that disappears after some time period.



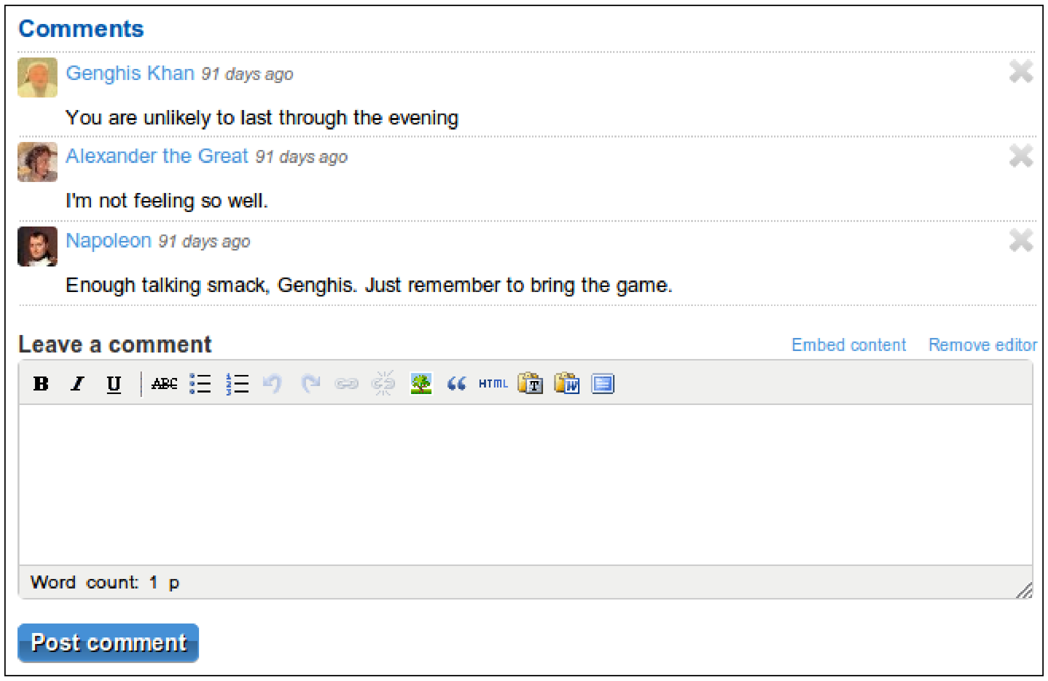
View: `page/elements/messages`

Developers: The messages are queued by the functions `system_message()` and `register_error()`.

Themers: Each message is an item in a list. The list has a class of `.elgg-system-messages` and the item's `.elgg-message`. An additional class on the item is used to distinguish between success and error messages (`.elgg-state-success` and `.elgg-state-error`).

Comments

The comments view provides a list of comments and an optional form for adding a new comment.

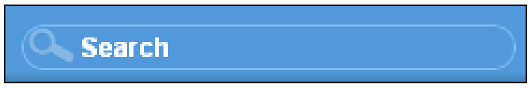


Views: `page/elements/comments`, `annotation/generic_comment`

Developers: Comments can be added to page through the convenience function `elgg_view_comments()`.

Search box

The search bar view is in the search plugin. It is added to the header by extending the `page/element/header view`**.**



View: `search/search_box`

Themers: The **Submit** button is hidden in the default theme using CSS.

Components

Just as software patterns exist, there are visual patterns found in website designs. Elgg has abstracted these patterns as reusable views paired with CSS markup. The view creates the HTML structure for a pattern. The CSS is divided between a base class that captures the shared styling and extension classes that describe the unique aspects of a particular instantiation of the pattern.

As an example, consider the module pattern which consists of a box with header and body:

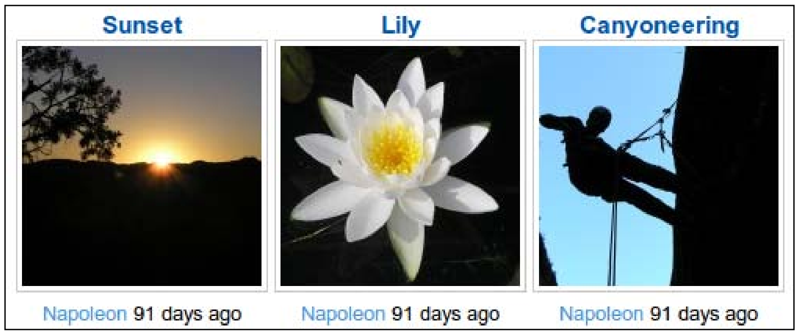
```
<div class="elgg-module elgg-module-featured">
  <div class="elgg-head">
    <h3>My Module</h3>
  </div>
  <div class="elgg-body">
    My content
  </div>
</div>
```

The base CSS class is `.elgg-module` and `.elgg-module-features` is the extension. The entire look and feel of the module can be changed by changing the extension class. This approach to HTML and CSS encourages reuse of markup leading to consistent HTML and smaller CSS files.

The implementation of these components was inspired by the **Object Oriented CSS Framework** created by Nicole Sullivan. There is a description of the approach behind this framework in *Chapter 9* of this book.

Gallery

The gallery pattern is popular on photo and e-commerce sites. It can be used interchangeably with the list pattern. The file plugin uses this view for displaying photos in a gallery.



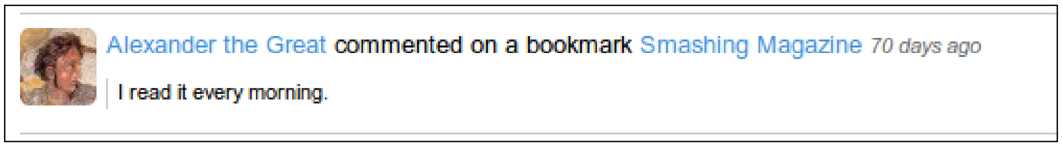
View: `page/components/gallery`

Developers: The gallery view requires fixed width content.

Themers: An inline-block approach is used on the unordered list to work around height differences in the list items. For more information on this, read `http://blog.mozilla.com/webdev/2009/02/20/cross-browser-inline-block/`.

Image block

The image block pattern appears on almost every web page in Elgg. It consists of an image on one side and a body of text on the other. The image is usually a user avatar or content icon. The image may appear on the left, right, or both sides of the body.



Alexander the Great commented on a bookmark Smashing Magazine *70 days ago*

I read it every morning.

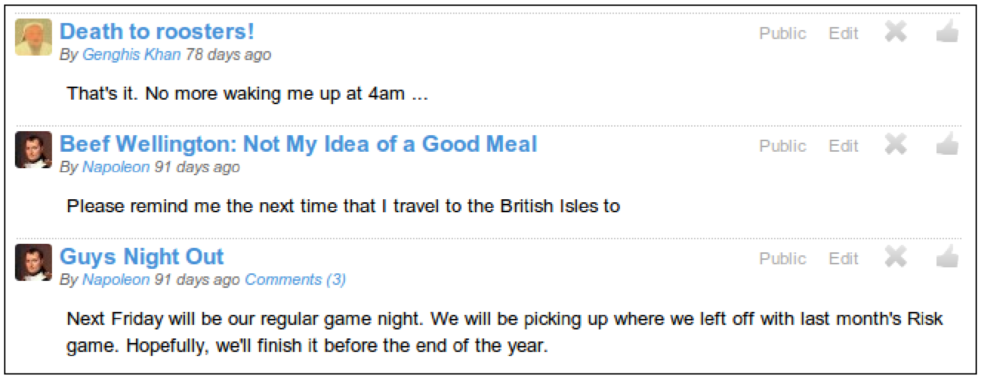
View: `page/components/image_block`

Developers: `elgg_view_image_block()` is a convenience function for rendering an image block.

Themers: The body of the image block uses a space-filling div using this technique: `http://www.stubbornella.org/content/2010/12/09/the-hacktastic-zoom-fix/`.

List

Lists appear often in Elgg. Activity streams, content lists, and comments all use the list component.



Death to roosters!
By Genghis Khan 78 days ago

That's it. No more waking me up at 4am ...

Beef Wellington: Not My Idea of a Good Meal
By Napoleon 91 days ago

Please remind me the next time that I travel to the British Isles to

Guys Night Out
By Napoleon 91 days ago Comments (3)

Next Friday will be our regular game night. We will be picking up where we left off with last month's Risk game. Hopefully, we'll finish it before the end of the year.

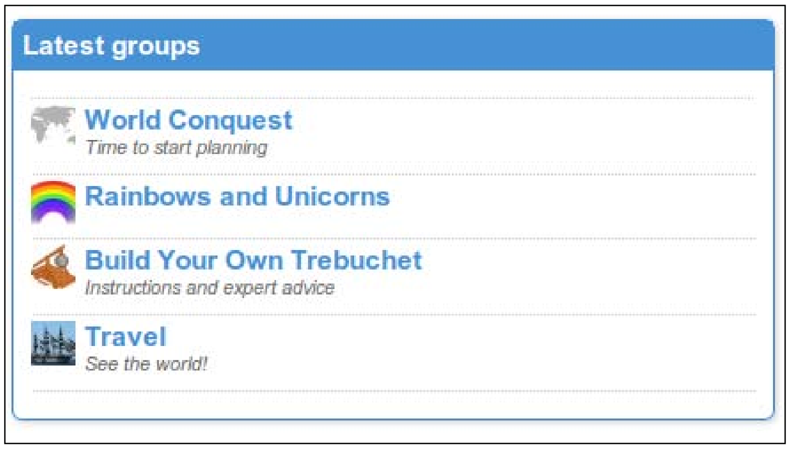
View: `page/components/list`

Developers: It is rare that this view is used directly in a plugin. Instead, the view is called by the many list convenience functions provided by Elgg such as `elgg_list_entities()`.

Themers: The list view, unsurprisingly, uses an HTML list element to structure the content. The base CSS class on the unordered list is `.elgg-list`, with each item having a class of `.elgg-item`.

Module

As already mentioned, a module consists of a box with a header and body. The module view also supports an optional footer.



View: `page/components/module`

Developers: `elgg_view_module()` is a convenience function for rendering a module.

Themers: The module does not support adding classes to the header, body, or footer. Instead, styling is accomplished through the extension classes on the module box, as follows:

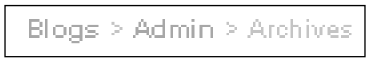
```
.elgg-module-feature > .elgg-head {
  border: 1px solid #4690D6;
}
```

Navigation

Elgg provides several types of navigation views. Themes may style some or all of these elements depending on how different the theme is from Elgg's default theme.

Breadcrumbs

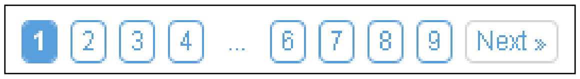
The breadcrumb view is used by plugins to indicate location in a page hierarchy. It works as a stack and plugin authors can add elements to the breadcrumbs by using `elgg_push_breadcrumb()`.



View: `navigation/breadcrumbs`

Pagination

This view provides paged navigation of content listings (blogs, files, search results). It is included automatically on the listing pages created with the `elgg_list*` functions.



View: `navigation/pagination`

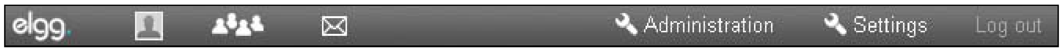
Menus

Elgg has more than 10 different menus that it natively supports (plugins can create additional menus). The menus all use an unordered list for HTML structure. The CSS classes are based on the menu name. Menus have a base CSS class, `.elgg-menu`, which is extended based on the menu name. For example, the CSS class for the site menu is `.elgg-menu-site`. Each section of a menu has its own class just as each individual menu item has a class. Finally, there are classes for hierarchical menus (`.elgg-menu-parent` and `.elgg-menu-child`). These classes make it easy to style and animate the menus without changing the HTML markup.

For information on adding or removing menu items, read the documentation included, `engine/lib/navigation.php` and the series of blog posts on the menu system published at `http://blog.elgg.org/pg/search/?q=menu&search_type=tags`.

Topbar menu

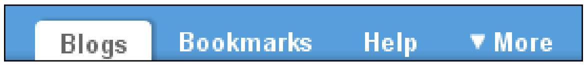
This menu has two sections ("`default`" and "`alt`").



View: `navigation/menu/default` (used by most menus)

Site menu

This menu is available on every page. To limit the width of the menu, Elgg registers a function, `elgg_site_menu_setup()`, to process its menu items before the menu is rendered. This function creates a "`default`" section and a "`more`" section as seen in the following screenshot:

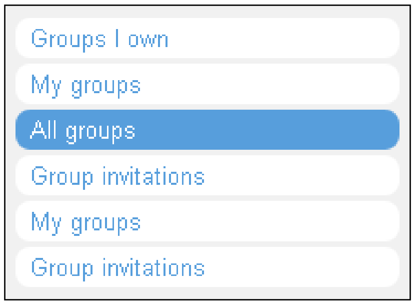


View: `navigation/menu/site`

Developers: To move this menu, remove it from the header by overriding that view and then insert it in a different location.

Page menu

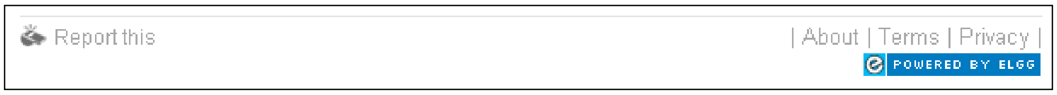
The page menu appears in the sidebar in the default theme. It has different menu items in different parts of the site because it uses context to determine what to display. The primary context of a web page is generally the first segment in its URL (`http://elgg.org/blog/all` has a context of "`blog`").



View: `navigation/menu/page`

Footer menu

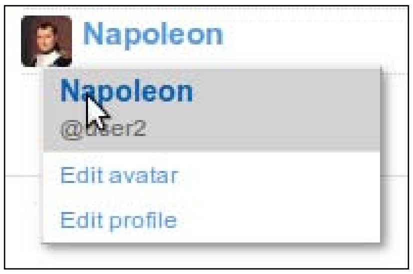
This menu is very similar to the topbar menu in that it uses the default menu view and has two sections ("`default`" and "`alt`").



View: `navigation/menu/default`

User hover menu

This is a dynamic menu. This means that its menu items are not registered until just before it is rendered. It has an "`actions`" section and an "`admin`" section.

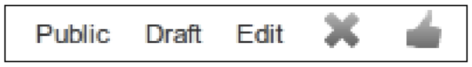


View: `navigation/menu/user_hover`

Comments: Do not use `elgg_register_menu_item()` to add an item to this menu. Instead, use the 'register', 'menu:user_hover' plugin hook.

Entity menu

The entity menu is used frequently when displaying lists of content, users, or groups. It gets its name from the fact that all of these are "`entities`" in Elgg's data model. It is also a dynamic menu.

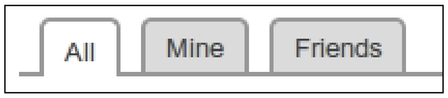


View: `navigation/menu/default`

Developers: Use the `'register', 'menu:entity'` plugin hook to add an item to this menu.

Tabs

Tabbed navigation is commonly used for selecting whose content to view. It is included on the main listing pages for the content plugins such as blogs or bookmarks.



View: `navigation/tabs`

Developers: This view does not handle the content of the tab panes, but only the navigation. Any plugin using it must manage the tab switching (through Ajax, hidden content or new page loads).

Themers: The tabs use an unordered list. The selected item uses the class `.elgg-state-selected`**.**

Forms

Elgg has views for collecting input from users and for displaying what was collected. Forms are the most common method for accepting user input. The form body consists of a set of input views (for textboxes, radio buttons, a submit button, and so on). It is highly recommended to use the input views for creating forms. This ensures the HTML markup is consistent across the site.

Generally, the data that was entered by the user will be displayed on the site. Elgg uses a parallel set of output views. These views add the appropriate markup for the data being displayed. URLs become hyperlinks, long chunks of text are automatically formatted in paragraphs, and time stamps are turned into human-readable date strings.

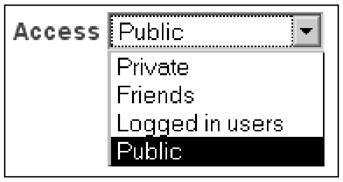
Input

Input views are primarily used in forms. They include buttons, textboxes, and checkboxes. Plugin authors are strongly encourages to these elements when building forms so that the HTML of forms is consistent.

At the top of each view file is a list of parameters that it accepts. All views accept a name, class, and id. The views also support all the attributes defined by the W3C standards for input elements. *Chapters 8* and *9* provide example usage of the input views.

Access

This control is used for selecting the access permissions on content. If access levels are not passed through the options variable, then it uses the access levels provided by `get_write_access_array()`. This function returns all the access levels available to the current user, including personal access collections.

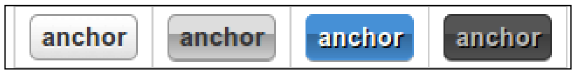


View: `input/access`

Buttons

Elgg has two primary button views. The "`input/submit`" view is used to add submit buttons to forms. The "`input/button`" view provides a general view for creating buttons.

There are CSS classes for creating submit, action, cancel, and delete buttons. In each case, a base class of `.elgg-button` is extended by a specific class (such as `.elgg-button-action` for an action button).

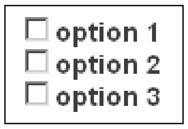


Views: `input/button`, `input/reset`, `input/submit`

Developers: The CSS classes can also be applied to anchor tags to create buttons from links.

Checkboxes

This view creates arrays of checkboxes. The labels and values are passed as an associative array.



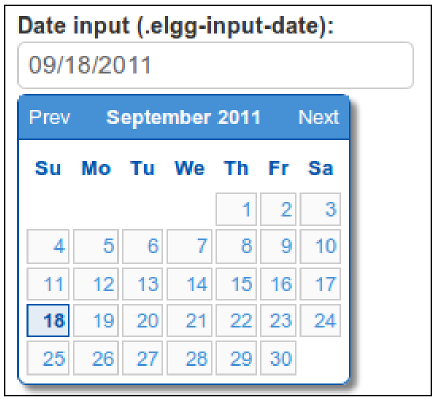
View: `input/checkboxes`

Developers: A hidden input field is added in this view. The hidden input has the same name as the checkbox with a value of 0. If a box is not checked, then 0 is passed to the action of the form. If at least one box is checked, then the value of the checkbox is sent. The values of checkboxes are submitted as an array. If a set of checkboxes is created with the name "`mycheckboxes`", then the first value is obtained in an action as follows:

```
$checkboxes = get_input('mycheckboxes', array());
$first_value = $checkboxes[0];
```

Date

The view for selecting dates displays a textbox. When a user clicks in the textbox, a calendar is displayed using JavaScript. When a day is selected using the calendar, that date is entered into the textbox.

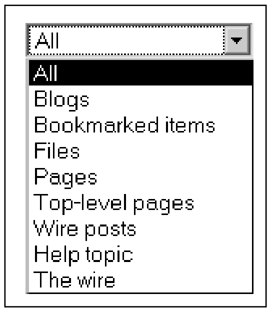


View: `input/date`

Themers: Elgg uses jQueryUI's date picker. Elgg's default theme includes custom CSS for the date picker. Themes can modify that CSS or pull CSS from a jQuery theme.

Drop-down selector

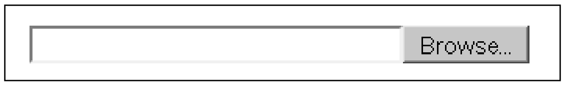
An associative array is used to set the value and label of the elements in this selector.



View: `input/dropdown`

File upload

The file upload view creates an input field of type file. It uses the web browser's file chooser to select a single file for upload. The form encoding type must be set to multipart/form-data to use this input field.



View: `input/file`

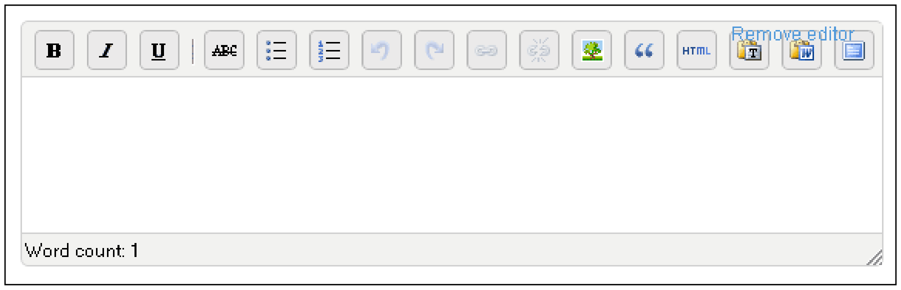
Hidden input

Use this to embed information into a form that should not be displayed. An example use is storing the identifier of a blog post in the comment form.

View: `input/hidden`

Large textarea

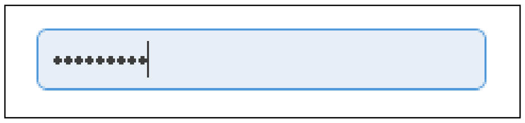
A large textarea element is available through two views: `input/longtext` and `input/plaintext`. The longtext view uses a WYSIWYG editor, if available, while the plain text view does not. The long text view also includes its own menu that can be extended.



Views: `input/longtext`, `input/plaintext`

Password

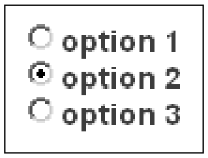
Password fields are created with the input/password view.



View: `input/password`

Radio buttons

This view creates an array of radio buttons with the same options as checkboxes. The key difference between the two is that only a single option can be selected from a set of radio buttons, as shown in the following screenshot:



View: `input/radio`

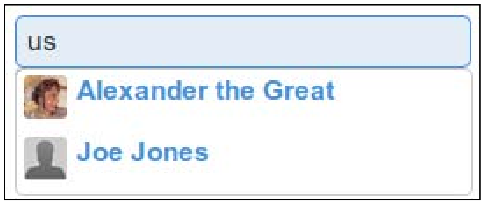
Textbox

There are several different textbox input views. They each have a paired output view that displays the data differently (for example, e-mail addresses turn into mailto links and tags become hyperlinks).

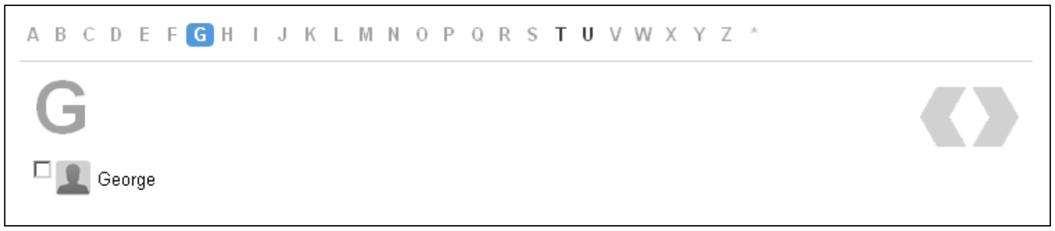
Views: `input/text`, `input/email`, `input/location`, `input/tags`, `input/url`

User pickers

There are two views for selecting users. The `input/userpicker` view uses an Ajax callback to display matching users as a name is typed.



The `input/friendspicker` view displays users alphabetically grouped. The access collection interface is an example usage of this view.



The `friendspicker` supports multiple selections and can be used to highlight previous selections.

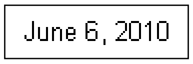
Views: `input/friendspicker`, `input/userpicker`

Output

There is generally a partner output view for every input view. These views assist in the display of information collected from users.

Date

The date output view accepts either a Unix time stamp or a date string with the output being a date string.



June 6, 2010

Views: `output/date`

E-mail address

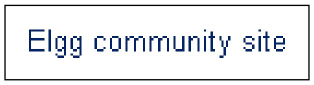
The e-mail output view turns an e-mail address into a `mailto` link.

cash.costello@gmail.com

View: `output/email`

Link

There are a few reasons to use the link output views provided by Elgg. First, they determine the full URL for a link from a segment of a URL (pass in `'blog/all'` and the view uses the URL `http://example.org/blog/all`). Second, the confirm link view pops up a dialog box to confirm any action the user is about to take such as deleting a blog post. Third, they can add security tokens to protect users (for information on this feature, visit `http://docs.elgg.org/Secure_forms`).

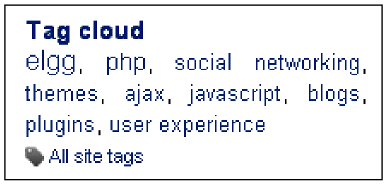


Elgg community site

Views: `output/url`, `output/confirmlink`

Tag cloud

The tag cloud view is called from `elgg_view_tagcloud()`. This function supports a wide range of parameters for determining what tags make up the cloud. The view sets the font size for each tag and includes a "`tagcloud/extend`" view for adding content at the bottom of the cloud.


Tag cloud
elgg, php, social networking, themes, ajax, javascript, blogs, plugins, user experience
All site tags

View: `output/tagcloud`

Tags

The tags view accepts an array of strings and returns a set of links for those tags. It is commonly used in the major plugins to list the tags attached to the content (such as tags on a file or a bookmark).

View: `output/tags`

Text

There are two primary text output views. The "`output/longtext`" view marks up the text with HTML to highlight links and format paragraphs. The other text view displays the text exactly as it was saved.

Views: `output/longtext`, `output/text`

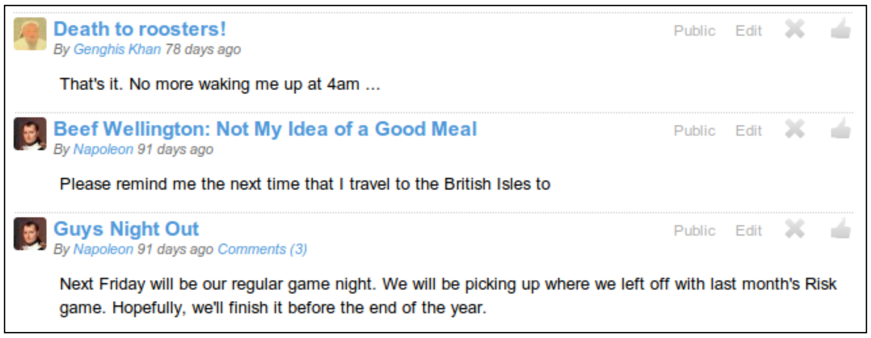
The form

Elgg has a convenience function called `elgg_view_form()` for rendering forms. If the Elgg action for the form is `'blog/save'`, then the form body view should be `'forms/blog/save'`. In that form body view, the labels and input elements are assembled. The `elgg_view_form()` function handles inserting the content into a form, setting the action, and including Elgg's security features for preventing Cross Site Request Forgeries (CSRF).

View: `input/form`

Users, groups, and objects

Appendix A, Developer's Quick Start Guide, discusses how users, groups, and objects (blogs, bookmarks, and files) are all entities in Elgg's data model. When rendering an entity, it can be done as a summary or as a full representation of the entity. The summary is used when listing several entities as in the list of blogs as shown in the following screenshot:


Death to roosters!
By Genghis Khan 78 days ago
That's it. No more waking me up at 4am ...
Beef Wellington: Not My Idea of a Good Meal
By Napoleon 91 days ago
Please remind me the next time that I travel to the British Isles to
Guys Night Out
By Napoleon 91 days ago Comments (3)
Next Friday will be our regular game night. We will be picking up where we left off with last month's Risk game. Hopefully, we'll finish it before the end of the year.

The summary uses the image block component described earlier. On the right is the entity menu that was mentioned in the menu section of this appendix. This menu has important information about the entity (such as its access level) and links for acting on the entity (liking or deleting the entity).

The full view of an entity usually has the summary at the top with its description plus comments below. Both the brief and full entity renderings occur through the same view. The name of the view depends on the type and subtype of the entity. If it is a blog, then the view is "`object/blog`". A bookmark uses "`object/bookmark`", a user "`user/default`", and a group "`group/default`".

Views: `object/elements/summary`, `object/*`, `user/*`, `group/*`

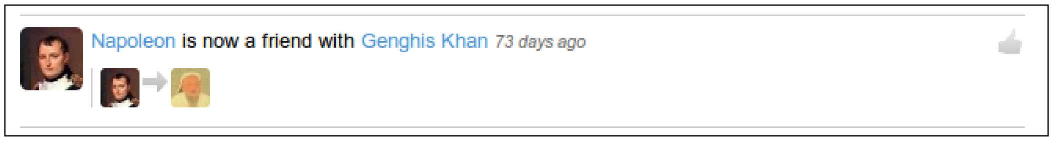
Developers: The function for rendering an entity is `elgg_view_entity()`.

Themers: The class `.elgg-output` is often used for the main content area when displaying an entity.

Activity stream

The activity stream, also called the river in Elgg, provides an overview of what is happening on a site. It uses the list pattern component. Each item is created with the image block pattern.

The body of each river item is made up of several views. There is a river menu for liking and toggling a response box. The summary is the text that describes the activity. There is also support for a message (an excerpt of text from the original content), an attachment (such as a thumbnail image), and a response (a comment box).



Napoleon is now a friend with Genghis Khan *73 days ago*

Views: `river/item`, `river/elements/*`

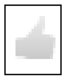
Developers: A plugin calls `add_to_river()` to include an event in the activity stream. The name of the view that creates the summary is passed into that function and usually based on the content type and the action. For example, posting a blog results in the view "`river/object/blog/create`" being used as set by the `blog/save` action.

Likes

The like system was added in Elgg 1.8 and is similar to Facebook's like button. This capability is provided by the likes plugin distributed with Elgg.

Like this

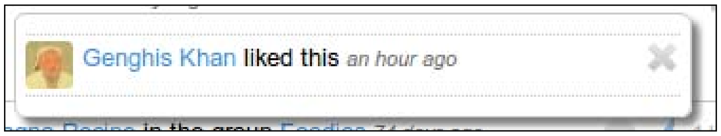
The button for liking something is a thumbs-up icon. It changes color based on whether the content has been liked by the viewer.



View: `likes/button`

List users

If at least one person has liked a piece of content, then a count of the likes is displayed next to the thumbs-up icon. Clicking on the count triggers a pop-up with a list of users that liked the content.



Genghis Khan liked this *an hour ago*

Views: `likes/count`, `annotation/likes`

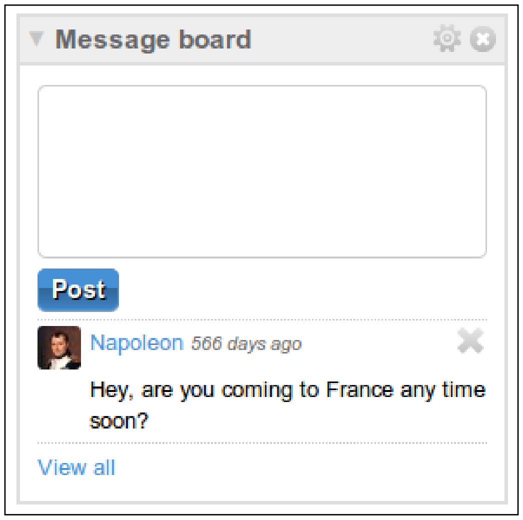
Themers: The thumbs-up icon is in Elgg's sprite. The icons have a base class of `.elgg-icon` that sets the background image and a class extension of `.elgg-icon-<icon name>` that sets the offset into the image.

Widgets

Elgg provides a widget framework including a layout with support for adding new widgets and views for rendering individual widgets. It is heavily dependent on the widget JavaScript library included with Elgg (based on jQuery UI).

The widget layout has a configurable number of columns. It also includes a button for toggling the display of a panel with the available widgets for that page.

An individual widget uses the module pattern. The body of the widget can include both the content of the widget and a settings box that can be toggled.



Napoleon 566 days ago

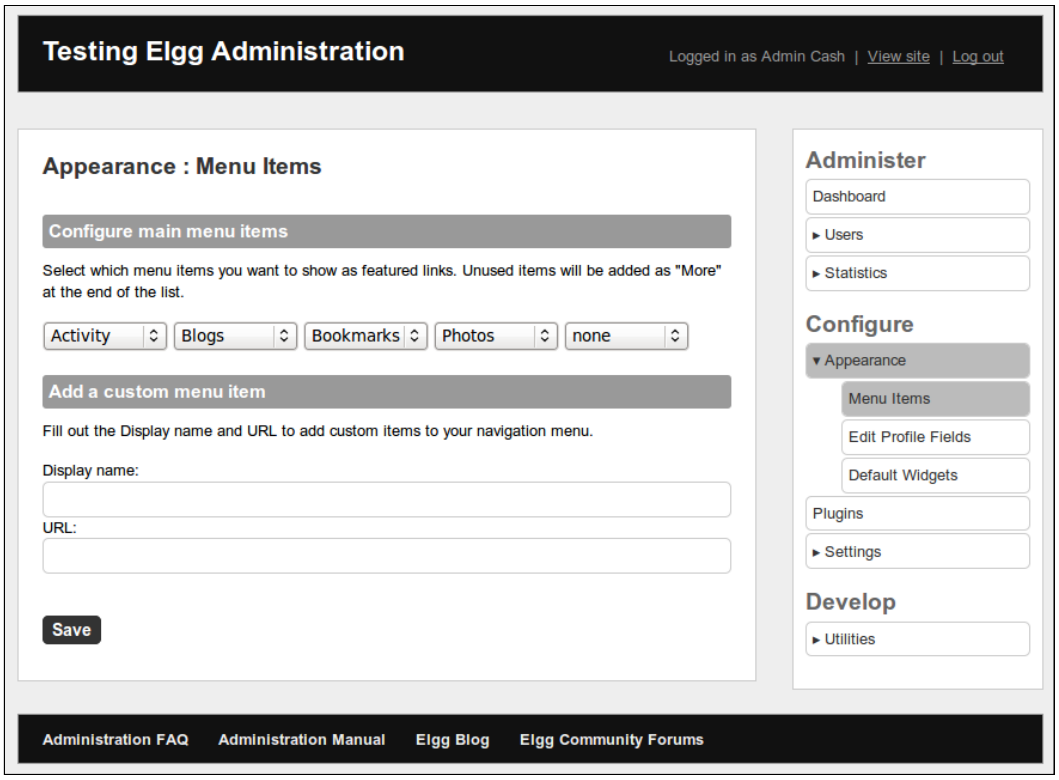
Hey, are you coming to France any time soon?

Views: `page/layout/widgets`, `object/widget`

Developers: *Lesson 5* in *Chapter 8* describes the process of creating a new widget.

Administration

The administrative backend of Elgg has a separate theme, giving it a consistent look and feel regardless of what is changing with the theme for the rest of the site. The views are found in the `admin` directory of the view tree.


Testing Elgg Administration

Logged in as Admin Cash | View site | Log out

Appearance : Menu Items

Configure main menu items

Select which menu items you want to show as featured links. Unused items will be added as "More" at the end of the list.

Activity | Blogs | Bookmarks | Photos | none

Add a custom menu item

Fill out the Display name and URL to add custom items to your navigation menu.

Display name:

URL:

Save

Views: `admin/*`, `page/admin`, `page/layout/admin`

Developers: Adding a new page to the admin area requires two steps:

1. Creating a view under the admin directory (for example, "`admin/configure/backup`").
2. Registering a link (`http://elgg.org/admin/configure/backup`) to the page menu based on the view name using the function `elgg_register_admin_menu_item()`.

Elgg handles creating the page and placing the content of that view into the layout.

Themers: The admin theme uses a fluid layout and could be used as a model for themes that do the same with the user-facing portion of the site.

Summary

The views catalog guided you through the use of many of the views provided by Elgg. This concludes this appendix and the book. Thank you for reading it and good luck with your Elgg endeavors.

Index

Thank you for buying

Elgg 1.8 Social Networking

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to author@packtpub.com. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
PUBLISHING community experience distilled

From Technologies to Solutions
PHP 5 e-commerce Development
Create a flexible framework in PHP for a powerful e-commerce solution
Michael Peacock
PACKT

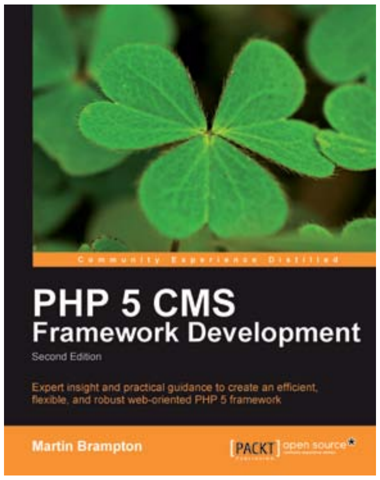
PHP 5 E-commerce Development

ISBN: 978-1-847199-64-5 Paperback: 356 pages

Create a flexible framework in PHP for a powerful e-commerce solution

1. Build a flexible e-commerce framework using PHP, which can be extended and modified for the purposes of any e-commerce site
2. Enable customer retention and more business by creating rich user experiences
3. Develop a suitable structure for your framework and create a registry to store core objects
4. Promote your e-commerce site using techniques with APIs such as Google Products or Amazon web services, SEO, marketing, and customer satisfaction

---


Community Experience Distilled
PHP 5 CMS Framework Development
Second Edition
Expert insight and practical guidance to create an efficient, flexible, and robust web-oriented PHP 5 framework
Martin Brampton
PACKT open source

PHP 5 CMS Framework Development - Second Edition

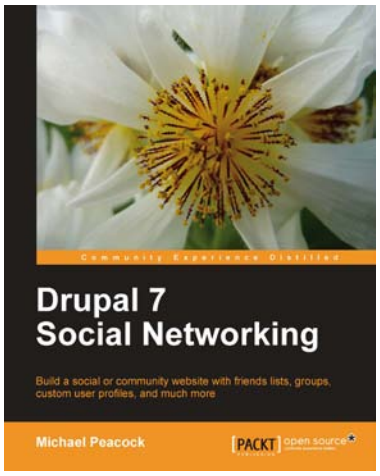
ISBN: 978-1-84951-134-6 Paperback: 416 pages

Expert insight and practical guidance to create an efficient, flexible, and robust web-oriented PHP 5 framework

1. Learn about the design choices involved in the creation of advanced web oriented PHP systems
2. Build an infrastructure for web applications that provides high functionality while avoiding pre-empting styling choices
3. Implement solid mechanisms for common features such as menus, presentation services, user management, and more

Please check **www.PacktPub.com** for information on our titles

[PACKT] open source
PUBLISHING
community experience distilled


Community Experience Distilled
Drupal 7 Social Networking
Build a social or community website with friends lists, groups, custom user profiles, and much more
Michael Peacock
PACKT open source

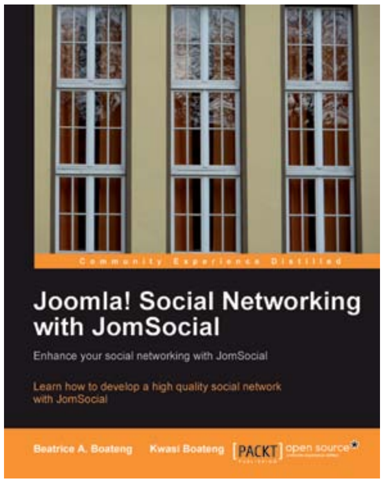
Drupal 7 Social Networking

ISBN: 978-1-84951-600-6 Paperback: 328 pages

Build a social or community website with friends lists, groups, custom user profiles, and much more

1. Step-by-step instructions for putting together a social networking site with Drupal 7
2. Customize your Drupal installation with modules and themes to match the needs of almost any social networking site
3. Allow users to collaborate and interact with each other on your site
4. Requires no prior knowledge of Drupal or PHP; but even experienced Drupal users will find this book useful to modify an existing installation into a social website

---


Community Experience Distilled
Joomla! Social Networking with JomSocial
Enhance your social networking with JomSocial
Learn how to develop a high quality social network with JomSocial
Beatrice A. Boateng Kwasi Boateng
PACKT open source

Joomla! Social Networking with JomSocial

ISBN: 978-1-847199-56-0 Paperback: 184 pages

Learn how to develop a high quality social network using JomSocial

1. Create and run your own social network with Joomla! and JomSocial
2. Creating content for the social network and integrating it with other Joomla! extensions
3. Community building and interactions

Please check **www.PacktPub.com** for information on our titles