Elgg Documentation

Versión master

Various

07 de diciembre de 2020

Índice general

Elgg (`pronunciation`) is a rapid development framework with built-in social features. It is a great fit for building any app where users log in and share information.

It has been used to build all kinds of social apps:

- open networks (similar to Facebook)
- topical (like the Elgg Community)
- private intranets
- dating
- educational
- company blog

This is the canonical documentation for the Elgg project.

CAPÍTULO 1

Primeros pasos

Descubra si Elgg es lo que necesita para su comunidad.

1.1 Funcionalidades

Showcases: https://elgg.org/showcase

1.1.1 Para desarrolladores

- Licencia permisiva.
- Sistema de temas.
- Internacionalización.
- Motor de modelos (templates).
- Sistema de artilugios (widgets).
- API de complementos.
- Grafo social.
- API de servicios web.
- Sistema de JavaScript basado en jQuery.
- Gestión de sesiones.
- Encaminamiento de direcciones URL personalizado.

1.1.2 Para administradores

- Perfiles y avatares de usuarios.

- Listas de control de acceso flexibles.
- Contactos y listas de contactos (similares a los círculos de G+).
- Diseño responsivo compatible con dispositivos móviles.
- Compatibilidad con RSS.
- Flujo de actividad.
- Complementos para tipos de contenido habituales, como blogs, marcadores, ficheros, microblogs, mensajes privados, documentos, tablones de mensajes y discusiones.
- Autenticación y administración de usuarios

Si necesita más funcionalidades de las que Elgg ofrece de manera predeterminada, dispone de varias opciones:

- Extienda las funcionalidades predeterminadas mediante el uso de *complementos*. Por ejemplo, para blogs, foros o marcadores sociales.
- Desarrolle sus propias funcionalidades mediante complementos.
- Contrate a alguien para que realice las tareas anteriores.

1.2 Bundled plugins

Elgg comes with a set of plugins. These provide the basic functionality for your social network.

1.2.1 Blog

A weblog, or blog, is arguably one of the fundamental DNA pieces of most types of social networking site. The simplest form of personal publishing, it allows for text-based notes to be published in reverse-chronological order. Commenting is also an important part of blogging, turning an individual act of publishing into a conversation.

Elgg's blog expands this model by providing per-entry access controls and cross-blog tagging. You can control exactly who can see each individual entry, as well as find other entries that people have written on similar topics. You can also see entries written by your friends (that you have access to).

Ver también:

Blogging on Wikipedia

1.2.2 Dashboard

The dashboard is bundled with both the full and core-only Elgg packages. This is a users portal to activity that is important to them both from within the site and from external sources. Using Elgg's powerful widget API, it is possible to build widgets that pull out relevant content from within an Elgg powered site as well as grab information from third party sources such as Twitter or Flickr (providing those widgets exist). A users dashboard is not the same as their profile, whereas the profile is for consumption by others, the dashboard is a space for users to use for their own needs.

Figura 1: A typical Elgg dashboard

1.2.3 Diagnostics

For the technically savvy user, system diagnostics enables you to quickly evaluate the server environment, Elgg code, and plugins of an Elgg install. Diagnostics is a core system plugin that comes turned on by default with Elgg. To download the diagnostics file, follow the steps below. The file is a dump of all sorts of useful information.

To use:

- Log in as Administrator
- Go to Administration -> Administer -> Utilities ->System Diagnostics
- Click "Download"

System diagnostics dump file contents:

- List of all Elgg files along with a hash for each file
- List of all the plugins
- PHP superglobals
- PHP settings
- Apache settings
- **Elgg CONFIG values**
    - language strings
    - site settings
    - database settings
    - plugin hooks
    - actions
    - views
    - page handlers
    - much more

1.2.4 File repository

The file repository allows users to upload any kind of file. As with everything in an Elgg system, you can filter uploaded files by tag and restrict access so that they're only visible by the people you want them to be. Each file may also have comments attached to it.

There are a number of different uses for this functionality

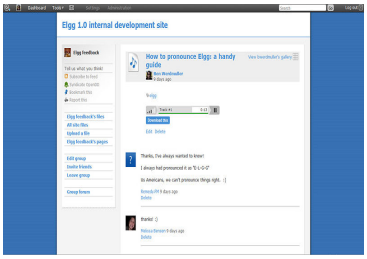

Figura 2: A file in an Elgg file repository

Photo gallery

When a user uploads photographs or other pictures, they are automatically collated into an Elgg photo gallery that can be browsed through. Users can also see pictures that their friends have uploaded, or see pictures attached to a group. Clicking into an individual file shows a larger version of the photo.

Podcasting

An Elgg file repository RSS feed automatically doubles as an RSS feed, so you can subscribe to new audio content using programs like iTunes.

Special content

It is possible for other plugins to add to the players available for different content types. It's possible for a plugin author to embed a viewer for Word documents, for example.

Note for developers

To add a special content type player, create a plugin with views of the form `file/specialcontent/mime/type`. For example, to create a special viewer for Word documents, you would create a view called `file/specialcontent/application/msword`, because `application/msword` is the MIME-type for Word documents. Within this view, the `ElggEntity` version of the file will be referenced as `$vars['entity']`. Therefore, the URL of the downloadable file is:

```
<?php echo $vars['url']; ?>action/file/download?file_guid=<?php echo $vars['entity']->
↪getGUID(); ?>
```

Using this, it should be possible to develop most types of embeddable viewers.

1.2.5 Groups

Once you have found others with similar interests - or perhaps you are part of a research groups or a course/class - you may want to have a more structured setting to share content and discuss ideas. This is where Elgg's powerful group building can be used. You can create and moderate as many groups as you like

- You can keep all group activity private to the group or you can use the "make public" option to disseminate work to the wider public.
- Each group produces granular RSS feeds, so it is easy to follow group developments
- Each group has its own URL and profile
- Each group comes with a *File repository*, forum, pages and messageboard

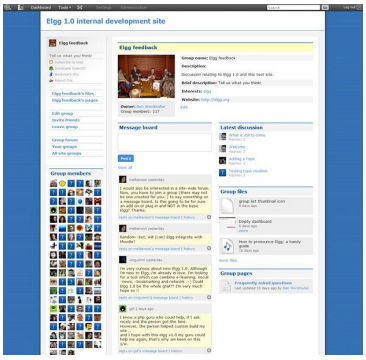

Figura 3: A typical group profile

1.2.6 Messageboard

The messageboard - similar to "The Wall" in Facebook or a comment wall in other networks is a plugin that lets users put a messageboard widget on their profile. Other users can then post messages that will appear on the messageboard. You can then reply directly to any message and view the history between yourself and the person posting the message.

1.2.7 Messages

Figura 5: Message notification

Private messaging can be sent to users by clicking on their avatar or profile link, providing you have permission. Then, using the built in *WYSIWYG editor*, it is possible to format the message. Each user has their own inbox and sentbox. It is possible to be notified via email of new messages.

When users first login, they will be notified about any new message by the messages notification mechanism in their top toolbar.

1.2.8 Pages

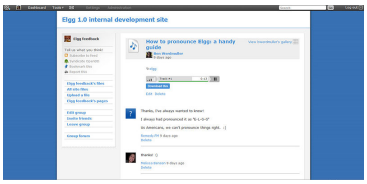

Figura 6: An Elgg Page

The pages plugin allows you to save and store hierarchically-organized pages of text, and restrict both reading and writing privileges to them. This means that you can collaboratively create a set of documents with a loose collection of people, participate in a writing process with a formal group, or simply use the functionality to write a document that only you can see, and only choose to share it once it's done. The easy navigation menu allows you to see the whole document structure from any page. You can create as many of these structures as you like; each individual page has its own access controls, so you can reveal portions of the structure while keeping others hidden. In keeping with all other elements in Elgg, you can add comments on a page, or search for pages by tag.

Usage

Pages really come into their own in two areas, firstly as a way for users to build up things such as a resume, reflective documentation and so on. The second thing is in the area of collaboration, especially when in the context of groups. With the powerful access controls on both read and write, this plugin is ideal for collaborative document creation.

Nota: Developers should note that there are actually 2 types of pages:

1. Top-level pages (with subtype `page_top`)
2. Normal pages (with subtype `page`)

1.2.9 Profile

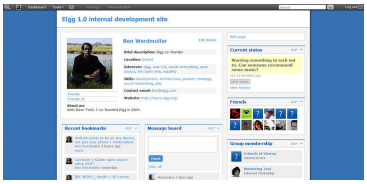

Figura 7: An Elgg profile

The profile plugin is bundled with both the full and core-only Elgg packages. The intention is that it can be disabled and replaced with another profile plugin if you wish. It provides a number of pieces of functionality which many consider fundamental to the concept of a social networking site, and is unique within the plugins because the profile icon it defines is referenced as standard from all over the system.

User details

This provides information about a user, which is configurable from within the plugin's start.php file. You can change the available profile fields form the admin panel. Each profile field has its own access restriction, so users can choose exactly who can see each individual element. Some of the fields contain tags (for example *skills*) limiting access to a field will also limit who can find you by that tag.

User avatar

The user avatar represents a user (or a group) throughout the site. By default, this includes a context-sensitive menu that allows you to perform actions on the user it belongs to wherever you see their avatar. For example, you can add them as a friend, send an internal message, and more. Each plugin can add to this context menu, so its full contents will vary depending on the functionality active in the current Elgg site.

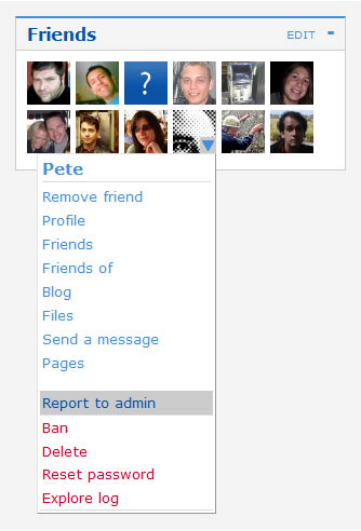

Figura 8: The Elgg context menu

Notes for developers

Using a different profile icon To replace the profile icon, or provide more content, extend the `icon/user/default` view.

Adding to the context menu The context menu can be expanded by registering a *plugin hook* for "register" "menu:user_hover", the following sections have special meaning:

- **default** for non-active links (eg to read a blog)
- **admin** for links accessible by administrators only

In each case, the user in question will be passed as `$params['entity']`.

1.2.10 The Wire

Elgg wire plugin «The Wire» is Twitter-style microblogging plugin that allows users to post notes to the wire.

The following plugins are also bundled with Elgg, but are not (yet) documented

- aalborg_theme
- bookmarks
- ckeditor
- custom_index
- developers
- embed
- externalpages
- garbagecollector
- htmlawed
- invitefriends
- legacy_urls
- likes
- logbrowser
- logrotate
- members
- notifications

- reportedcontent
- search
- site_notifications
- tagcloud
- twitter_api
- uservalidationbyemail
- web_services

1.3 Licencia

1.3.1 MIT o GPLv2

Existe un paquete completo de Elgg, con la infraestructura y un grupo de complementos fundamentales, disponible bajo los términos de la versión 2 de la Licencia Pública General de GNU (GPLv2). La infraestructura (sin los complementos) está además disponible bajo los términos de la licencia MIT.

1.3.2 Preguntas frecuentes

Las siguientes respuestas se facilitan por su comodidad, y no constituyen asesoramiento legal. Consulte las respuestas con un abogado para mayor seguridad. La Elgg Foundation no se hace responsable de decisiones tomadas en base al contenido de esta página.

Para obtener respuestas a cuestiones no planteadas en esta página, consulte las preguntas más habituales oficiales de la licencia GPLv2.

¿Cuánto cuesta Elgg?

Elgg puede descargarse, instalarse y usarse de manera gratuita. Si desea realizar una donación, le agradeceremos su apoyo económico!

Puedo eliminar el logotipo y los enlaces sobre Elgg?

Sí

¿Puedo modificar el código fuente?

Sí, pero en general recomendamos que realice sus cambios como un plugin, así cuando una nueva version de Elgg sea liberada, el proceso de actualización será menos difícil

¿Puedo cobrar a mis usuarios una cuota de pertenencia?

Sí

Si yo modifico Elgg, ¿ tengo que hacer mis cambios disponibles para todos?

Non, si está utilizando Elgg para ofrecer un servicio, no tiene por qué publicar el código fuente. Si redistribuye una versión modificada de Elgg, entonces sí debe incluir el código fuente con sus cambios.

Si utilizo Elgg para alojar una red, tiene la Elgg Foundation algún tipo de derecho sobre mi red?

No

¿Cuál es la diferencia entre la versión MIT y la GPL?

La versión bajo los términos de la licencia MIT no incluye ningún complemento.

Puede distribuir un producto comercial basado en Elgg usando la versión MIT, y no publicar sus modificaciones.

Con la versión bajo los términos de la licencia GPL, tiene la obligación de publicar cualquier cambio que realice sobre la infraestructura, en caso de redistribuir la misma.

¿Por qué no hay complementos en la versión MIT?

Los complementos se desarrollaron bajo los términos de la licencia GPL, por lo que no es posible publicarlos bajo los términos de la licencia MIT. Además, algunos complementos dependen de otros componentes que tampoco son compatibles con la licencia MIT.

¿Puedo distribuir un complemento para Elgg utilizando una licencia privativa?

Creemos que sí, puesto que los complementos suelen depender únicamente de la infraestructura, que está disponible bajo los términos de la licencia MIT. Dicho esto, le recomendamos encarecidamente que lo consulte con un abogado para estar completamente seguro.

Tenga en cuenta que los complementos que se publican en el repositorio de la comunidad deben estar bajo los términos de licencias compatibles con la GPLv2. No tienen por que usar necesariamente la licencia GPLv2, sino que pueden usar otra licencia compatible, como la MIT.

¿Podemos crear nuestra propia herramienta que use Elgg y vendérsela a nuestros clientes?

Sí, pero sus clientes podrán redistribuir dicha herramienta bajo los términos de la licencia GPLv2.

1.4 Instalación

Ponga a andar su propia instancia de Elgg en un momento.

Contents

1.4.1 Requisitos

- MySQL 5+
- PHP 5.6+ with the following extensions:
  - GD (para el procesamiento de imágenes).
  - PDO (for database connection)
  - JSON (for AJAX responses, etc.)
  - XML (for reading plugin manifest files, etc.)
  - Compatibilidad con cadenas de tamaño variable (para internacionalización).
  - Configuración adecuada y posibilidad de enviar mensajes de correo electrónico mediante un agente de transferencia de mensajes (MTA).
- Servidor web que permita reescribir direcciones URL.

Official support is provided for the following configurations:

- **Apache server**
  - Apache con el módulo rewrite module activado.
  - PHP ejecutado como un módulo de Apache.
- **Nginx server**
  - Nginx with PHP-FPM using FastCGI

Por «compatibilidad oficial» se entiende:

- Most development and testing is performed with these configurations
- Much of the installation documentation is written assuming Apache or Nginx is used
- Priority on bug reports is given to Apache and Nginx users if the bug is web server specific (but those are rare).

Browser support policy

Feature branches support the latest 2 versions of all major browsers as were available at the time of the first stable release on that branch.

Bugfix release will not alter browser support, even if a new version of the browser has since been released.

Major browsers here means all of the following, plus their mobile counterparts:

- Android Browser
- Chrome
- Firefox
- IE
- Safari

«Support» may mean that we take advantage of newer, unimplemented technologies but provide a JavaScript polyfill for the browsers that need it.

You may find that Elgg happens to work on unsupported browsers, but compatibility may break at any time, even during a bugfix release.

1.4.2 Resumen

Envíe Elgg

With Composer (recommended if comfortable with CLI):

```
cd /path/to/wwwroot/
composer self-update
composer global require "fxp/composer-asset-plugin:~1.3"
composer create-project elgg/starter-project:dev-master .
composer install
composer install # 2nd call is currently required
```

From pre-packaged zip (recommended if not comfortable with CLI):

- Descargue la última versión de Elgg.
- Envíe el archivo ZIP a su servidor con un cliente de FTP.
- Unzip the files in your domain's document root.

Cree una carpeta de datos

Elgg necesita una carpeta especial en la que almacenar los ficheros enviados al sitio, incluidos los iconos de perfil y las fotos de los usuarios. Tendrá que crear una carpeta para este fin.

Atención: For security reasons, this folder **MUST** be stored outside of your document root. If you created it under `/www/` or `/public_html/`, you're doing it wrong.

Una vez la carpeta está creada, tendrá que asegurarse de que el servidor web en el que se está ejecutando Elgg tiene permisos para escribir y crear subcarpetas en esa carpeta. No debería ser un problema en servidores Windows, pero si su servidor está en un GNU/Linux, Mac OS X o en una variante de UNIX, tendrá que establecer los permisos en la carpeta.

Si está enviando los ficheros mediante un cliente de FTP gráfico, es posible que el editor le permita cambiar los permisos haciendo clic derecho en la carpeta y seleccionando «Propiedades» o «Obtener información».

Nota: Directories must be executable to be read and written to. The suggested permissions depend upon the exact server and user configuration. If the data directory is owned by the web server user, the recommended permissions are `750`.

Advertencia: Setting your data directory to `777` will work, but it is insecure and is not recommended. If you are unsure how to correctly set permissions, contact your host for more information.

Cree una base de datos en MySQL

Cree una nueva base de datos en MySQL para Elgg usando la herramienta de administración de bases de datos que prefiera (pregunte al administrador del sistema si tiene alguna duda).

Asegúrese de que añade un usuario a la base de datos con todos los privilegios y de que se queda con el nombre de la base de datos, del usuario y de la contraseña. Necesitará esta información durante la instalación de Elgg.

Set up Cron

Elgg uses timed requests to your site to perform background tasks like sending notifications or performing database cleanup jobs. You need to configure the *cron* to be able to use those kind of features.

Visite su sitio Elgg

Una vez completados los pasos, visite su sitio Elgg desde un navegador web. Elgg le presentará un asistente que le guiará a través del proceso de instalación. La primera cuenta que cree al finalizar la instalación será una cuenta de administrador.

Una nota sobre settings.php y .htaccess

El instalador de Elgg intentará crear dos ficheros por usted:

- `elgg-config/settings.php`, which contains local environment configuration for your installation
- `.htaccess`, which allows Elgg to generate dynamic URLs

Si los ficheros no se pueden generar automáticamente, por ejemplo porque el servidor carezca de permisos de escritura en las carteras correspondientes, Elgg le ofrecerá instrucciones para crearlos usted manualmente. También puede cambiar los permisos de la carpeta raíz y de la carpeta del motor de manera temporal. Defina los permisos en esas dos carpetas de manera que permitan que el servidor escriba esos dos ficheros, complete el proceso de instalación, y vuelva a cambiar los permisos a su valor previo. Si por algún motivo esto no funcionase, necesitará:

- In `elgg-config/`, copy `settings.example.php` to `settings.php`, open it up in a text editor and fill in your database details
- On Apache server, copy `install/config/htaccess.dist` to `.htaccess`
- On Nginx server copy `install/config/nginx.dist` to `/etc/nginx/sites-enabled` and adjust it's contents

1.4.3 Otras configuraciones

- Cloud9
- Homestead
- EasyPHP
- IIS
- MAMP
- MariaDB
- Nginx
- Ubuntu
- Virtual hosts
- XAMPP

1.4.4 Solución de problemas

¡Ayuda! Tengo problemas al instalar Elgg

En primer lugar:

- Asegúrese de que el servidor cumple con los requisitos técnicos de Elgg.
- Siga las instrucciones aquí definidas para su entorno de ser necesario.
- ¿Se ha asegurado de que `mod_rewrite` está cargado?
- ¿Se está cargando el módulo de MySQL para Apache?

Tome nota de los pasos que siga para arreglar la instalación. A veces, al cambiar algunas opciones o ficheros para intentar solucionar el problema, es posible que cause otros problemas más adelante. Si necesita empezar de cero, sólo tiene que eliminar todos los ficheros, eliminar la base de datos, y volver a empezar.

No puedo guardar la configuración durante la instalación, me sale un error 404 al intentarlo

Elgg utiliza la extensión de Apache `mod_rewrite` para simular ciertas direcciones URL. Por ejemplo, cada vez que realiza una acción en Elgg, o cuando visita el perfil de un usuario, el servidor traduce internamente la URL a algo que Elgg comprende. Esto se hace mediante reglas definidas en el fichero `.htaccess`, que es la manera estándar de definir configuración adicional de un sitio en Apache.

Este error sugiere que las reglas de `mod_rewrite` no se están aplicando correctamente. Los motivos pueden ser varios. Si no se siente cómodo o no puede aplicar las soluciones que le presentamos a continuación, le recomendamos encarecidamente que contacte con el administrador de sistemas o asistente técnico correspondiente y le haga llegar esta página a él.

The `.htaccess`, if not generated automatically (that happens when you have problem with `mod_rewrite`), you can create it by renaming `install/config/htaccess.dist` file you find with elgg package to `.htaccess`. Also if you find a `.htaccess` file inside the installation path, but you are still getting 404 error, make sure the contents of `.htaccess` are same as that of `install/config/htaccess.dist`.

‘‘mod_rewrite‘‘ no está instalado.

Revise el fichero de configuración de Apache, `httpd.conf`, y asegúrese de que el módulo `mod_rewrite` se está cargando. Si ha necesitado cambiar la configuración, acuérdese de reiniciar Apache para que los cambios surtan efecto. También puede usar PHP info para comprobar si el módulo está cargado.

No se están respetando las reglas del fichero ‘‘.htaccess‘‘.

En la configuración de su servidor virtual (virtual host), que puede formar parte de `httpd.conf`, cambie el valor de «AllowOverride» de forma que quede como sigue:

```
AllowOverride all
```

Esto le indicará a Apache que debe utilizar las reglas de substitución (`mod_rewrite`) del fichero `.htaccess`.

Elgg no está instalador en la carpeta raíz del servidor web. Por ejemplo, está instalado en «http://example.org/elgg/» en vez de en «http://example.org/».

El script de instalación me redirige a «action» en vez de a «actions»

This is a problem with your `mod_rewrite` setup.

Atención: DO NOT, REPEAT, DO NOT change any directory names!

Instalé en un subdirectorio y mi instalación no funciona !

Si usted instaló Elgg, este será accedido con una dirección como http://example.org/mysite/ (http://ejemplo.org/misitio/) en lugar de http://example.org/ (http://ejemplo.org), hay una pequeña probabilidad de que las «rewrte rules» o reglas de reescritura, del fichero .htaccess no lo procesen correctamente. Esto normalmente es debido al uso de alias con Apache. Usted puede necesitar dar un puntero a mod_rewrite que apunte a donde su instalación de Elgg se encuentra.

- Abra el fichero .htaccess en un editor de texto
- Donde le señale, añada una línea como ‘‘RewriteBase /path/to/your/elgg/installation/‘ (RewriteBase/camino/hacia/su/ellg/instalacion/) No olvide las barras finales.
- Salve el fichero y recargue su navegador

Por favor, dése cuenta que el camino que está usando es el “web”, quitándole la parte del host.

Por ejemplo, si la instalación de Elgg se encuentra en http://example.org/elgg/, deberá usar la siguiente base:

```
RewriteBase /elgg/
```

Por favor, dése cuenta que instalar en un subdirectorio no requiere el uso de «RewriteBase». Esto solo es necesario en las raras circunstancias en las que la configuración del servidor lo requiere.

Lo hice todo! mod_rewrite está trabajando bien, pero sin embargo el error 404 sigue apareciendo.

Es posible que haya algún tipo de problema con el fichero «.htaccess». A veces la rutina de instalación de Elgg no es capaz de crear dicho fichero y tampoco es capaz de avisar de que no ha podido crearlo. Si ha probado ya todo lo anterior, llegados a este punto:

- Compruebe si el fichero «.htaccess» es realmente el de Elgg, y no uno básico instalado por el propio servidor.
- Si efectivamente no es el fichero de Elgg, use «htaccess_dist» (cámbiele el nombre a «.htaccess»).

Recibo un mensaje de error diciendo que el test de reescritura ha fallado después de chequear la pagina de requerimientos

Me aparecen estos mensajes tras el paso de comprobación de los requisitos (el segundo paso) de la instalación:

> Creemos que su servidor se encuentra ejecutando el web server Apache.
>
> No se ha superado la prueba de substitución, probablemente porque la opción «AllowOverride» no tiene el valor «All» para la carpeta de Elgg. Esto impide que Apache procese el fichero «htaccess» que define las reglas de substitución.
>
> Otra posible causa, aunque menos probable, es que Apache esté configurado con un alias para la carpeta de Elgg, y que por ello necesite definir la opción «RewriteBase» en el fichero «.htaccess». Encontrará más información al respecto en el fichero «.htaccess» de la carpeta de Elgg.

Tras recibir este error, cualquier interacción con la interfaz web arroja un error 500 (error interno del servidor).

Lo más probable es que la causa sea el no haber cargado el módulo de filtros eliminando el prefijo de comentario de la línea:

> #LoadModule filter_module modules/mod_filter.so

del fichero de configuración de Apache «httpd.conf».

El fichero «error.log» de Apache contendrá una entrada como la siguiente:

> . . . .htaccess: Orden incorrecta: «AddOutputFilterByType», es posible que no esté bien escrita o que esté definida en un módulo que no está incluido en la configuración del servidor.

Hay una página en blanco después de poner los parámetros de mi base de datos

Asegúrese de que el módulo de MySQL de Apache está instalado y se está cargando.

Estoy teniendo un error 404 con una url muy larga

Si obtiene un error 404 durante la instalación o durante la creación del primero de los usuarios con una URL como «http://example.com/homepages/26/d147515119/htdocs/elgg/action/register», significa que la URL del sitio definida en la tabla sites_entity de la base de datos es incorrecta. Elgg intenta adivinar el valor, pero en el caso de servidores compartidos no siempre acierta. Use phpMyAdmin para editar el valor en la base de datos y cambiarlo a la URL correcta.

Estoy teniendo problemas poniendo mi camino de datos

This is highly server specific so it is difficult to give specific advice. If you have created a directory for uploading data, make sure your http server can access it. The easiest (but least secure) way to do this is give it permissions `777`. It is better to give the web server ownership of the directory and limit the permissions.

> **Advertencia:** Setting directory permissions to `777` allows the **ENTIRE** internet to place files in your directory structure an possibly infect you webserver with malware. Setting permissions to `750` should be more than enough.

La causa principal de este problema es que PHP esté configurado para evitar el acceso a la mayor parte de las carpetas mediante la opción open_basedir. Consulte a su proveedor de servidor.

Asegúrese de que la ruta es correcta y de que termina en «/». Puede comprobar la ruta en la base de datos, en la tabla «datalists».

Si sólo puede acceder a su servidor mediante FTP y ha creado una carpeta pero desconoce su ruta completa, quizá pueda descubrir cuál es a partir de la ruta del fichero «www» definida en la tabla «datalists» de la base de datos. Llegados a este punto, se recomienda pedir ayuda al equipo de asistencia técnica de su servicio de servidores.

No puedo validar mi cuenta de administración ya que no tengo un servidor de correo

Si bien es cierto que las cuentas normales (sin contar las creadas desde el panel de administración) deben tener asociada y autenticada una cuenta de correo electrónico para poder usarse, la cuenta de administrador lo necesita.

Una vez registrada la primera cuenta, podrá acceder al sitio con los datos de acceso que haya indicado durante el registro de la cuenta.

Lo he intentado todo, y sigo sin poder instalar Elgg

Es posible que, durante el proceso de depuración de su problema de instalación original, haya causado algún otro problema. Pruebe a realizar una instalación desde cero de nuevo.

- Pare su base de datos Elgg

- Borre su directorio de datos
- Borre los ficheros fuente de Elgg
- Empiece de nuevo.

Si aún así no es capaz de instalar Elgg, busque la ayuda de la comunidad de Elgg. Asegúrese de indicar la versión de Elgg que está intentando instalar, así como detalles sobre la plataforma del servidor, y cualquier mensaje de error que reciba, incluyendo aquellos que contenga el historial (log) del servidor.

1.5 Developer Overview

This is a quick developer introduction to Elgg. It covers the basic approach to working with Elgg as a framework, and mentions some of the terms and technologies used.

See the *Guías para desarrolladores* for tutorials or the *Documentos de diseño* for in-depth discussion on design.

1.5.1 Database and Persistence

Elgg uses MySQL 5.5 or higher for data persistence, and maps database values into Entities (a representation of an atomic unit of information) and Extenders (additional information and descriptions about Entities). Elgg supports additional information such as relationships between Entities, activity streams, and various types of settings.

1.5.2 Plugins

Plugins change the behavior or appearance of Elgg by overriding views, or by handling events and plugin hooks. All changes to an Elgg site should be implemented through plugins to ensure upgrading core is easy.

1.5.3 Actions

Actions are the primary way users interact with an Elgg site. Actions are registered by plugins.

1.5.4 Events and Plugin Hooks

Events and Plugin Hooks are used in Elgg Plugins to interact with the Elgg engine under certain circumstances. Events and hooks are triggered at strategic times throughout Elgg's boot and execution process, and allows plugins to modify or cancel the default behavior.

1.5.5 Views

Views are the primary presentation layer for Elgg. Views can be overridden or extended by Plugins. Views are categories into a Viewtype, which hints at what sort of output should be expected by the view.

1.5.6 JavaScript

Elgg uses an AMD-compatible JavaScript system provided by RequireJs. Bundled with Elgg are jQuery, jQuery UI, jQuery Form, jQuery jeditable, and jQuery UI Autocomplete.

Plugins can load their own JS libs.

1.5.7 Internacionalización.

Elgg's interface supports multiple languages, and uses Transifex for translation.

1.5.8 Caching

Elgg uses two caches to improve performance: a system cache and SimpleCache.

1.5.9 3rd party libraries

The use of 3rd party libraries in Elgg is managed by using Composer dependencies. Examples of 3rd party libraries are jQuery, RequireJs or Zend mail.

To get a list of all the Elgg dependencies check out the Packagist page for Elgg.

1.5.10 Database Seeding

Elgg provides some base database seeds to populate the database with entities for testing purposes.

You can run the following commands to seed and unseed the database.

..code::sh

    # seed the database composer database:seeder:seed

    # unseed the database composer database:seeder:unseed

CAPÍTULO 2

Guías para administradores

Mejores prácticas para gestionar un sitio basado en Elgg de manera efectiva.

2.1 Getting Started

You have installed Elgg and worked through any potential initial issues. What now? Here are some suggestions on how to to familiarize yourself with Elgg.

2.1.1 Focus first on core functionality

When you're new to Elgg, it's best to explore the stock features in core and its bundled plugins before installing any third party plugins. It's tempting install every interesting plugin from the community site, but exploring the core features builds a familiarity with Elgg's expected behavior, and prevents introducing any confusing bugs from third party plugin into your new Elgg network.

Elgg installs with a basic set of social network plugins activated: blogs, social bookmarking, files, groups, likes, message boards, wiki-like pages, user profiles, and microblogging. To change the plugins that are activated, log in as an admin user, then use the topbar to browse to Administration, then to Plugins on the right sidebar.

Nota: The user you create during installation is an admin user.

2.1.2 Create test users

Users can be created two ways in stock Elgg:

1. Complete the signup process using a different email address and username. (Logout first or use a different browser!)
2. Add a user through the Admin section by browsing to Administration -> Users -> Add New User.

Nota: Users that self-register must validate their account through email before they can log in. Users that an admin creates are already validated.

2.1.3 Explore user functionality

Use your test users to create blogs, add widgets to your profile or dashboard, post to the Wire (microblogging), and create pages (wiki-like page creation). Investigate the Settings on the topbar. This is where a user sets notification settings and configures tools (which will be blank because none of the default plugins add controls here).

2.1.4 Explore admin functionality

All of the admin controls are found by clicking Administration in the topbar. The has a dashboard with a widget that explains the various sections. Change options in the Configure menu to change how Elgg looks and acts.

2.1.5 Extending Elgg

After exploring what Elgg can do out of the box, install some themes and plugins. You can find many plugins and themes at the community site that have been developed by third parties. These plugins do everything from changing language strings, to adding chat, to completely redesigning Elgg's interface. Because these plugins are not official, be certain to check the comments to make sure you only install well-written plugins by high quality developers.

2.2 Actualizar Elgg

Actualice un sitio existente a una nueva versión de Elgg.

Si ha escrito complementos personalizados, debería leer también las guías para desarrolladores para así *obtener información sobre cómo actualizar el código de esos complementos </guides/upgrading>* para adaptarlo a la última versión de Elgg.

2.2.1 Advice

- **Back up your database** and code
- Mind any version-specific comments below
- Upgrade only one minor version at a time (1.6 => 1.7, then 1.7 => 1.8)
- Pruebe la nueva versión en un sitio de prueba antes de realizar la actualización.
- Report any problems in plugins to the plugin authors
- If you are a plugin author you can report any backwards-compatibility issues to GitHub

2.2.2 Basic instructions

1. Log in as an admin to your site
2. Disable caching in Advanced Settings
3. **Back up your database, data directory, and code**

4. Download the new version of Elgg from http://elgg.org
5. **Update the files**
    - If doing a patch upgrade (1.9.x), overwrite your existing files with the new version of Elgg
    - If doing a minor upgrade (1.x), replace the existing core files completely
6. **Merge any new changes to the rewrite rules**
    - For Apache from `install/config/htaccess.dist` into `.htaccess`
    - For Nginx from `install/config/nginx.dist` into your server configuration (usually inside `/etc/nginx/sites-enabled`)
7. Merge any new changes from `settings.example.php` into `settings.php`
8. Visit http://your-elgg-site.com/upgrade.php

Nota: Cualquier modificación de Elgg debería estar hecha mediante complementos, de forma que no se pierda nada al substituir la instalación de Elgg actual por una nueva. Si no es este el caso, preocúpese de mantener sus propias modificaciones.

2.2.3 From 2.2 to 2.3

PHP Version

PHP 5.5 has reached end of life in July 2016. To ensure that Elgg sites are secure, we now require PHP 5.6 for new installations.

Existing installations can continue using PHP 5.5 until Elgg 3.0.

In order to upgrade Elgg to 2.3 using composer while using PHP 5.5, you may need to use `--ignore-platform-reqs` flag.

Tests

- PHPUnit bootstrap is deprecated by composer autoloader: Tests should no longer bootstrap themselves using `/engine/tests/phpunit/bootstrap.php`. Instead, tests should extend `\Elgg\TestCase`.
- Some core files now sniff if `PHPUNIT_ELGG_TESTING_APPLICATION` constant is set to determine whether Elgg is being bootstrapped for PHPUnit tests. `phpunit.xml` configuration needs to updated to include this constant definition.
- PHPUnit bootstrap no longer sets global `$CONFIG`. Tests should use `_elgg_services()->config` instead.
- Core and tests no longer use private global values in `$_ELGG->view_path` and `$_ELGG->allowed_ajax_views`

Schema

- The database GUID columns need to be aligned. In the admin section an upgrade is available to handle this. Please make sure you have a backup available

2.2.4 From 1.x to 2.0

Removed plugins

The following plugins are no longer bundled with Elgg core:

- categories (https://github.com/elgg/categories)
- zaudio (https://github.com/elgg/zaudio)

IE-specific workarounds have been dropped

Several views (`css/ie`, `css/ie7`, `css/ie8`, etc.) as well as conditional comments have been discarded now that IE10+ browsers are more standards-compliant. If you need browser support farther back than that, you will need to find or build a plugin that introduces its own compatibility layer or polyfills.

Update your webserver config

URL paths like `cache/*` and `rewrite.php` now use the main front controller script. You **must** remove these rewrite rules from your webserver config (e.g. `.htaccess`).

Also remove the rules for paths like `export/*`; these endpoints have been removed.

Settings location

After upgrading, move your `settings.php` file from `engine/` to `elgg-config/`.

2.2.5 From 1.10 to 1.11

Breaking changes

In versions 1.9 and 1.10, names and values for metadata and annotations were not correctly trimmed for whitespace. Elgg 1.11 correctly trims these strings and updates the database to correct existing strings. If your plugin uses metadata or annotations with leading or trailing whitespace, you will need to update the plugin to trim the names and values. This is especially important if you are using custom SQL clauses or have hard-coded metastring IDs, since the update might change metastring IDs.

2.2.6 De la versión 1.8 a la 1.9

Elgg 1.9 is a much lighter upgrade than 1.8 was.

Breaking changes

Plugins and themes written for 1.8 are expected to be compatible with 1.9 except as it pertains to comments, discussion replies, and notifications. Please report any backwards compatibility issues besides those just listed.

Upgrade steps

There are several data migrations involved, so it is especially important that you **back up your database and data directory** before performing the upgrade.

Download the new version and copy these files from the existing 1.8 site:

- `.htaccess`
- `engine/settings.php`
- any 3rd-party plugin folders in the `mod` directory

Then replace the old installation directory with the new one. This way you are guaranteed to get rid of obsolete files which might cause problems if left behind.

Follow the basic instructions listed above.

After you've visited `upgrade.php`, go to the admin area of your site. You should see a notification that you have pending upgrades. Click the link in the notification bar to view and run the upgrades.

The new notifications system delivers messages via a minutely cron handler. If you haven't done so yet, you will need to *install and configure crontab* on your server. If cron jobs are already configured, note that the scope of available cron periods may have changed and you may need to update your current crontab to reflect these changes.

Time commitment

Running all of the listed upgrades took about 1 hour and 15 minutes on the Elgg community site which at the time had to migrate:

- ~75,000 discussion replies
- ~75,000 comments
- ~75,000 data directories

You should take this only as a ballpark estimate for your own upgrade. How long it takes will depend on how large your site is and how powerful your servers are.

2.2.7 De la versión 1.7 a la 1.8

La versión 1.8 ha sido el mayor salto en el desarrollo de Elgg desde la versión 1.0. Es por ello que para actualizar el núcleo de Elgg y sus complementos hace falta más trabajo que en anteriores actualizaciones.

Actualizar el núcleo

Elimine las siguientes carpetas (se encuentran en el mismo nivel que «_graphics» o «engine»):

- _css
- account
- admin
- dashboard
- entities
- friends
- search

- settings
- simplecache
- views

Advertencia: Tendrá problemas si no elimina estas carpetas antes de actualizar.

2.3 Complementos

Los complementos pueden modificar el comportamiento de Elgg y añadir nuevas funcionalidades.

Contents

2.3.1 Dónde conseguir complementos

Puede obtener complementos de:

- The Elgg Community
- Github
- Sitios de terceros (normalmente por un precio).

If no existing plugins meet your needs, you can hire a developer or *create your own*.

2.3.2 The Elgg Community

Finding Plugins

Sort based on most popular

On the community plugin page, you can sort by date uploaded (Filter: Newest) or number of downloads (Filter: Most downloads). Sorting by the number of downloads is a good idea if you are new to Elgg and want to see which plugins

are frequently used by other administrators. These will often (but not always) be higher quality plugins that provide significant capabilities.

Use the plugin tag search

Next to the filtering control on the plugin page is a search box. It enables you to search by tags. Plugins authors choose the tags.

Look for particular plugin authors

The quality of plugins varies substantially. If you find a plugin that works well on your site, you can check what else that plugin author has developed by clicking on their name when viewing a plugin.

Evaluating Plugins

Look at the comments and ratings

Before downloading and using a plugin, it is always a good idea to read through the comments that others have left. If you see people complaining that the plugin does not work or makes their site unstable, you probably want to stay away from that plugin. The caveat to that is that sometimes users ignore installation instructions or incorrectly install a plugin and then leave negative feedback. Further, some plugin authors have chosen to not allow comments.

Install on a test site

If you are trying out a plugin for the first time, it is a bad idea to install it on your production site. You should maintain a separate test site for evaluating plugins. It is a good idea to slowly roll out new plugins to your production site even after they pass your evaluation on your test site. This enables you to isolate problems introduced by a new plugin.

2.3.3 Tipos de complementos

Temas

Themes are plugins that modify the look-and-feel of your site. They generally include stylesheets, client-side scripts and views that alter the default presentation and behavior of Elgg.

Paquetes de idioma

Language packs are plugins that provide support for other languages.

Language packs can extend and include translations for language strings found in the core, core plugins and/or third-party plugins.

Some of the language packs are already included in the core, and can be found in `languages` directory off Elgg's root directory. Individual plugins tend to include their translations under the `languages` directory within the plugin's root.

This structure makes it easy to create new language packs that supercede existing language strings or add support for new languages.

2.3.4 Instalación

All plugins reside in the `mod` directory of your Elgg installation.

To install a new plugin:

- extract (unzip) contents of the plugin distribution package
- copy/FTP the extracted folder into the `mod` directory of your Elgg installation, making sure that `manifest.xml` and `start.php` are directly under the plugin directory (e.g. if you were to install a plugin called `my_elgg_plugin`, plugin's manifest would need to be found at `mod/my_elgg_plugin/manifest.xml`)
- activate the plugin from your admin panel

To activate a plugin:

- Acceda al sitio con su cuenta de administrador.
- Vaya a «Administración → Configurar → Complementos».
- Localice su complemento en la lista, y haga clic en el botón «Activar».

2.3.5 Plugin order

Plugins are loaded according to the order they are listed on the Plugins page. The initial ordering after an install is more or less random. As more plugins are added by an administrator, they are placed at the bottom of the list.

Some general rules for ordering plugins:

- A theme plugin should be last or at least near the bottom
- A plugin that modifies the behavior of another plugin should be lower in the plugin list

2.3.6 Notas sobre versiones anteriores a la 1.8

En la versión 1.7 de Elgg y versiones anteriores, la interfaz para gestionar los complementos instalados está situada en «Administración → Administración de herramientas».

2.4 Rendimiento

Optimice la velocidad de respuesta del sitio.

Contents

2.4.1 ¿Puede Elgg funcionar con X millones de usuarios?

La gente suele preguntar si Elgg es capaz de funcionar con una gran cantidad de usuarios.

En primer lugar, podríamos preguntar «¿de dónde esperar sacar a todos esos usuarios?». Pero, bromas aparte, se trata de un problema muy interesante. Hacer que Elgg permita un gran número de usuarios es un problema de ingeniería técnica. Es un problema interesante, pero más o menos resuelto. La ciencia de la computación no funciona de manera diferente para Elgg de lo que lo hace para Google, por ejemplo. ¿Conseguir millones de usuarios? Eso es como el Santo Grial de toda la industria tecnológica.

En segundo lugar, como para casi todo en esta vida, la respuesta es «depende»:

- ¿Cómo de activos son los usuarios?
- ¿Qué hardware está utilizando en el servidor de Elgg?
- ¿Están comportándose correctamente los complementos?

Mejorar la eficiencia del motor de Elgg es un proyecto en progreso, pero existen límites en lo que puede hacer un script.

Si de verdad le preocupa el problema de la escalabilidad, le interesará echarle una ojeada a una serie de cosas por su cuenta.

2.4.2 Mida antes de nada

No sirve de nada emplear recursos para resolver un problema si desconoce:

- Cuál es el problema.
- Qué recursos necesita para solucionarlo.
- Dónde necesita situar dichos recursos.

Invierta en algún tipo de análisis que le ayude a descubrir dónde se encuentra su cuello de botella, especialmente si está pensando en invertir una cantidad de dinero importante en solucionar un problema.

2.4.3 Ajuste la configuración de MySQL

Elgg hace un gran uso de la base de datos, realizando varias consultas por cada vez que se carga una página. Esto es perfectamente normal, y un servidor de bases de datos correctamente configurado debería poder lidiar con millares de solicitudes por segundo.

A continuación le ofrecemos algunos consejos que podrían servirle:

- Asegúrese de que MySQL está configurado para usar el fichero de configuración (my.cnf) apropiado para las dimensiones de su sitio.
- Aumente la cantidad de memoria disponible para PHP y MySQL. En cualquiera de los casos necesitará aumentar la cantidad de memoria disponible para el proceso de PHP.

2.4.4 Active un sistema de caché

Por regla general, si un programa es lento, es porque está realizando alguna operación de computación costosa de manera repetitiva. Un sistema de caché permite al sistema evitar realizar ese trabajo una y otra vez, pues el resultado de la operación se almacena en memoria de forma que no sea necesario realizar el procesamiento de nuevo en las llamadas siguientes. A continuación se plantean algunas soluciones de caché disponibles que puede utilizar para Elgg.

Caché simple

De manera predeterminada, las vistas se guardan en una caché en la carpeta de datos de Elgg durante un cierto período de tiempo. Esto elimina la necesidad de regenerar una vista cada vez que se carga una página.

Puede desactivar esta caché con `$CONFIG->simplecache_enabled = false;`. Para mejorar la eficiencia del sitio, asegúrese de que el valor de esta variable es `true`.

La caché puede resultar problemática durante el proceso de desarrollo, concretamente durante la edición de temas en su complemento, pues usar la versión de la caché tendrá preferencia sobre usar la que ofrece su complemento.

La caché simple puede desactivarse desde el menú de administración. Se recomienda que la desactive durante el desarrollo de la plataforma si está programando complementos para Elgg.

Esta caché se elimina de manera automática al activar, desactivar o cambiar la posición de un complemento. También cuando se ejecuta `upgrade.php`.

For best performance, you can also create a symlink from `/cache/` in your www root dir to the `/views_simplecache/` directory in the data directory you configured when you installed Elgg:

```
cd /path/to/wwwroot/
ln -s /path/to/dataroot/views_simplecache/ cache
```

If your webserver supports following symlinks, this will serve files straight off disk without booting up PHP each time.

For security reasons, some webservers (e.g. Apache in version 2.4) might follow the symlinks by default only if the owner of the symlink source and target match. If the cache symlink fails to work on your server, you can change the owner of the cache symlink itself (and not the `/views_simplecache/` directory) with

```
cd /path/to/wwwroot/
chown -h wwwrun:www cache
```

In this example it's assumed that the `/views_simplecache/` directory in the data directory is owned by the wwwrun account that belongs to the www group. If this is not the case on your server, you have to modify the chown command accordingly.

Caché del sistema

El lugar en el que se encuentran las vistas se guarda en caché para que no haga falta descubrirlo cada vez (un análisis determinó que la carga de páginas requería una cantidad de tiempo que aumentaba de manera no lineal a medida que se añadían componentes, debido al sistema de descubrimiento de vistas). Elgg también guarda en caché información como la asociación (mapping) de idiomas o el mapa de clases.

Puede desactivar esta caché con `$CONFIG->system_cache_enabled = false;`. Para mejorar la eficiencia del sitio, asegúrese de que el valor de esta variable es `true`.

Actualmente esta información se almacena en la carpeta de datos, aunque versiones futuras de Elgg podrían almacenarla utilizando Memcache. Al igual que con la caché simple, ésta se elimina al activar, desactivar o cambiar la posición de un complemento, así como cuando se ejecuta `upgrade.php`.

La caché del sistema puede desactivarse desde el menú de administración. Se recomienda que la desactive durante el desarrollo de la plataforma si está programando complementos para Elgg.

Boot cache (experimental)

Elgg has the ability to cache numerous resources created and fetched during the boot process. To enable this cache you must set a TTL in your `settings.php` file: `$CONFIG->boot_cache_ttl = 10;`

A small TTL is recommended because it brings all the benefits of caching under load while reducing the harm if Elgg's cache invalidation strategy should miss something.

Caché de consultas

Durante el tiempo de carga de una página concreta, se almacenan en caché los resultados de todas las consultas `SELECT` que realiza la página. Esto significa que cada consulta distinta que una página realice durante la carga de una página sólo se realizará una vez, incluso en el caso de que la carga de la página efectúe la consulta en varias ocasiones. Cualquier operación de escritura en la base de datos elimina esta caché, por lo que se recomienda que, en páginas complejas, se pospongan las operaciones de escritura en la base de datos hasta el final de la carga de la página, o que se use la funcionalidad `execute_delayed_*`. Esta caché se elimina de manera automática en cuanto termina la carga de la página.

Puede que sufra problemas de memoria si usa la infraestructura de Elgg como una biblioteca en un script de la línea de órdenes en PHP. Para evitar estos problemas, desactive esta caché mediante `$CONFIG->db_disable_query_cache = true;`.

Etags y cabeceras de caducidad

Estas tecnologías informan a los navegadores web de los usuarios de cuándo guardar contenido estático (CSS, JavaScript, imágenes) en la caché local. Activar estas tecnologías reduce de manera considerable la carga del servidor y mejora el rendimiento desde el punto de vista del usuario.

Use el complemento de yslow para Firefox o «Chrome DevTools Audits» para confirmar qué tecnologías su sitio está utilizando actualmente.

Si los contenidos estáticos no se están guardando en caché:

- Asegúrese de que tiene estas extensiones instaladas y activadas en el servidor.
- Actualice el fichero «.htaccess» si está actualizando Elgg desde una versión anterior.
- Active la *caché simple*, que convierte las vistas seleccionadas en contenido que los navegadores web pueden guardar en caché.

Memcache

Memcache es una tecnología de caché genérica desarrollada por Brad Fitzpatrick para LiveJournal.

Advertencia: La compatibilidad con Memcache está en fase experimental, y es susceptible de cambiar.

Requisitos para su instalación:

- php5-memcache
- memcached

Configuración:

Uncomment and populate the following sections in `settings.php`

```
$CONFIG->memcache = true;

$CONFIG->memcache_servers = array (
    array('server1', 11211),
    array('server2', 11211)
);
```

Optionaly if you run multiple Elgg installations but use ony one Memcache server, you may want to add a namespace prefix. In order to do this, uncomment the following line

```
$CONFIG->memcache_namespace_prefix = '';
```

Squid

Hemos obtenido buenos resultados usando Squid para mantener una caché de las imágenes.

Caché de código

Existen varios sistemas de caché de código PHP intermedio (bytecode) en el mercado. Estos sistemas aumentan el rendimiento de su sitio utilizando cachés para el código intermedio compilado a partir de su script, de forma que el servidor no tiene que compilar el código PHP cada vez que ejecuta éste.

Direct file serving

If your server can be configured to support the X-Sendfile or X-Accel headers, you can configure it to be used in `settings.php`. This allows your web server to directly stream files to the client instead of using PHP's `readfile()`.

2.4.5 Servidor

No pretenda ofrecer un sitio capaz de alojar a millones de usuarios si sólo dispone de un servidor compartido de bajo coste. Necesitará un servidor propio y dedicado, con acceso a la configuración, así como un buen ancho de banda y gran cantidad de memoria.

Memoria, procesador y ancho de banda

Debido a cómo funcionan las cachés, los sistemas de caché requerirán memoria. Solucionar los problemas aumentando la memoria o mejorando el procesador suele salir barato.

En hardware avanzado, el cuello de botella será probablemente el ancho de banda del que dispone el propio servidor. Asegúrese de que la conexión que tiene contratada es suficiente para la demanda que espera que tenga el sitio.

Configuración

Finalmente, échele un ojo a su configuración, pues hay algunos aspectos inesperados que pueden darle problemas.

Por ejemplo, de manera predeterminada, Apache puede manejar una carga de trabajo alta. Sin embargo, la mayoría de las distribuciones de GNU/Linux vienen con un MySQL configurado para sitios pequeños. Esto puede llevar a que los procesos de Apache se queden pasados a la espera de comunicarse con alguno de los procesos de un MySQL sobrecargado.

2.4.6 Revise los complementos que no se estén comportando adecuadamente

Los complementos pueden programarse sin tener en cuenta el rendimiento, y un único complemento puede llegar a ralentizar todo el sitio.

Pruebe a desactivar algunos complementos y observar si ello mejora el rendimiento de manera notable. Una vez haya detectado algún complemento que parezca el responsable de los problemas de rendimiento, contacte con el autor original del complemento e infórmelo de sus hallazgos.

2.4.7 Use HTML generado por el cliente

Hemos descubierto que, llegados a cierto punto, una gran parte del tiempo empleado del lado del servidor consiste en generar el HTML de la página con el sistema de vistas de Elgg.

Es muy difícil guardar en caché la salida de los modelos (templates), dado que generalmente pueden rellenarse con una infinidad de datos distintos. En vez de intentar guardar en caché la salida HTML de ciertas páginas o vistas, le sugerimos que se pase a un sistema de modelos de páginas y vistas basada en HTML, de forma de que navegador web del usuario pueda guardar los propios modelos en caché. Entonces, deje que el ordenador del usuario realice el trabajo de generar el documento final aplicando datos en formato JSON a esos modelos.

Este sistema puede resultar muy efectivo, pero tiene la pega de que requiere unos costes de desarrollo adicionales significantes. El equipo de Elgg está considerando la posibilidad de integrar esta estrategia en el propio Elgg de manera directa dada su gran efectividad, especialmente en páginas con contenido repetido u oculto.

2.5 Cron

Cron is a program available on Unix-based operating systems that enables users to run commands and scripts at set intervals or at specific times.

Elgg's cron handler allows administrators and plugin developers to setup jobs that need to be executed at set intervals.

Most common examples of cron jobs in Elgg include:

- sending out queued notifications
- rotating the system log in the database
- collecting garbage in the database (compacting the database by removing entries that are no longer required)

Currently, Elgg supports the following hooks:

- `minute` - Run every minute
- `fiveminute` - Run every 5 minutes
- `fifteenmin` - Run every 15 minutes
- `halfhour` - Run every 30 minutes
- `hourly` - Run every hour
- `daily` - Run every day
- `weekly` - Run every week
- `monthly` - Run every month
- `yearly` - Run every year

Nota: `reboot` cron hook has been deprecated and should not be used

2.5.1 How does it work?

Elgg activates its cron handler when particular cron pages are loaded. As an example, loading http://example.com/cron/hourly/ in a web browser activates the hourly hook. To automate this, cron jobs are setup to hit those pages at certain times. This is done by setting up a `crontab` which is a configuration file that determines what cron jobs do and at what interval.

2.5.2 Instalación

The `crontab` needs to specify a script or command that will hit the Elgg cron pages. Two commonly available programs for this are *GET* and *wget*. You will need to determine the location of one of these on your server. Your crontab also needs to specify the location of your website.

```
# Crontab example.
#
# This file is an example of triggering Elgg cron events. It hits a URL to
# trigger the events. For testing, you can simulate the cronjob by loading the
# URL in a browser.
#
# See http://learn.elgg.org/en/stable/admin/cron.html for more information
#

# Location of your site (don't forget the trailing slash!)
ELGG='http://www.example.com/'

# Location of lwp-request
LWPR='/usr/bin/lwp-request'

# Make GET request and discard content
GET="$LWPR -m GET -d"

# The crontab
# Don't edit below this line unless you know what you are doing
* * * * * $GET ${ELGG}cron/minute/
```

(continué en la próxima página)

(proviene de la página anterior)

```
*/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/
# reboot is deprecated and probably doesn't work
@reboot $GET ${ELGG}cron/reboot/
```

In the above example, change the `ELGG` and `GET` variables to match you server setup. If you have SSH access to your Linux servers, type `crontab -e` and add your crontab configuration. If you already have a crontab configured, you will have to merge Elgg information into it. If you don't have SSH access, you will have to use a web-based configuration tool. This will vary depending on hosting provider.

If you choose the `wget` utility, you might want to consider these flags:

- `--output-document` or `-O` to specify the location of the concatenated output file. For example, under Debian: `/usr/bin/wget --output-document=/dev/null`. If you don't do that, a new file will be created for each cron page load in the home directory of the cron user.
- `--spider` to prevent the cron page from being downloaded.

On Windows servers, there is a number of cron emulators available.

For information on setting up cron jobs using cPanel see cPanel Docs.

In the `command` field, enter the appropriate link of the cron page. For example, for a weekly cron job, enter the command as http://www.example.com/cron/weekly/.

To see if your cron jobs are running, visit Statistics > Cron in your Elgg admin panel.

2.6 Backup and Restore

Contents

2.6.1 Introduction

Why

Shared hosting providers typically don't provide an automated way to backup your Elgg installation. This article will address a method of accomplishing this task.

In IT there are often many ways to accomplish the same thing. Keep that in mind. This article will explain one method to backup and restore your Elgg installation on a shared hosting provider that uses the CPanel application. However, the ideas presented here can be tailored to other applications as well. The following are typical situations that might require a procedure such as this:

- Disaster Recovery
- Moving your Elgg site to a new host
- Duplicating an installation

What

Topics covered:

- Full backups of the Elgg directories and MySQL databases are performed daily (automated)
- The backups are sent to an off-site location via FTP (automated)
- The local backups are deleted after successful transfer to the off-site location (automatic)
- Five days of backups will be maintained (automated)
- Restoration of data to the new host (manual)

This process was composed with assistance from previous articles in the Elgg documentation wiki.

Assumptions

The following assumptions have been made:

- The Elgg program directory is `/home/userx/public_html`
- The Elgg data directory is `/home/userx/elggdata`
- You've created a local directory for your backups at `/home/userx/sitebackups`
- You have an off-site FTP server to send the backup files to

- The directory that you will be saving the off-site backups to is `/home/usery/sitebackups/`
- You will be restoring the site to a second shared hosting provider in the `/home/usery/public_html` directory

---

Importante: Be sure to replace `userx`, `usery`, `http://mynewdomain.com` and all passwords with values that reflect your actual installation!

2.6.2 Creating a usable backup - automatically

Customize the backup script

The script that you will use can be found *here* .

Just copy the script to a text file and name the file with a .pl extension. You can use any text editor to update the file.

Change the following to reflect your directory structure:

```
# ENTER THE PATH TO THE DIRECTORY YOU WANT TO BACKUP, NO TRAILING SLASH
$directory_to_backup = '/home/userx/public_html';
$directory_to_backup2 = '/home/userx/elggdata';
# ENTER THE PATH TO THE DIRECTORY YOU WISH TO SAVE THE BACKUP FILE TO, NO TRAILING␣
↪SLASH
$backup_dest_dir = '/home/userx/sitebackups';
```

Change the following to reflect your database parameters:

```
# MYSQL BACKUP PARAMETERS
$dbhost = 'localhost';
$dbuser = 'userx_elgg';
$dbpwd = 'dbpassword';
# ENTER DATABASE NAME
$database_names_elgg = 'userx_elgg';
```

Change the following to reflect your off-site FTP server parameters:

```
# FTP PARAMETERS
$ftp_host = "FTP HOSTNAME/IP";
$ftp_user = "ftpuser";
$ftp_pwd = "ftppassword";
$ftp_dir = "/";
```

Save the file with the `.pl` extension (for the purposes of this article we will name the file: `elgg-ftp-backup-script.pl`) and upload it to the following directory `/home/userx/sitebackups`

Be aware that you can turn off FTP and flip a bit in the script so that it does not delete the local backup file in the event that you don't want to use off-site storage for your backups.

Configure the backup Cron job

Login to your CPanel application and click on the «Cron Jobs» link. In the Common Settings dropdown choose «Once a day» and type the following in the command field `/usr/bin/perl /home/userx/sitebackups/elgg-ftp-backup-script.pl`

Click on the «Add New Cron Job» button. Daily full backups are now scheduled and will be transferred off-site.

Configure the cleanup Cron job

If you are sending your backups, via FTP, to another shared hosting provider that uses the CPanel application or you've turned off FTP altogether you can configure your data retention as follows.

Login to your CPanel application for your FTP site, or locally if you're not using FTP, and click on the «Cron Jobs» link. In the Common Settings dropdown choose «Once a day» and type the following in the command field `find /home/usery/sitebackups/full_* -mtime +4 -exec rm {} \;`

The `-mtime X` parameter will set the number of days to retain backups. All files older than `x` number of days will be deleted. Click on the «Add New Cron Job» button. You have now configured your backup retention time.

2.6.3 Restoring from backup

Prepare your backup files

The assumption is that you're restoring your site to another shared hosting provider with CPanel.

When the script backed the files up the original directory structure was maintained in the zip file. We need to do a little cleanup. Perform the following:

- Download the backup file that you wish to restore from
- Extract the contents of the backup file
- **Drill down and you will find your site backup and SQL backup. Extract both of these. You will then have:**
    - a MySQL dump file with a `.sql` extension
    - **another directory structure with the contents of:**
        - `/home/userx/public_html`
        - `/home/userx/elggdata`
- **Repackage the contents of the `/home/userx/public_html` directory as a zip file so that the files are in the root of the**
    - The reason for doing this is simple. It's much more efficient to upload one zip file than it is to ftp the contents of the `/home/userx/public_html` directory to your new host.
- Repackage the contents of the /home/userx/elggdata directory as a zip file so that the files are in the root of the zip file

You should now have the following files:

- the `.sql` file
- the zip file with the contents of `/home/userx/public_html` in the root
- the zip file with the contents of `/home/userx/elggdata` in the root

Restore the files

This is written with the assumption that you're restoring to a different host but maintaining the original directory structure. Perform the following:

- Login to the CPanel application on the host that you wish to restore the site to and open the File Manager.
- **Navigate to `/home/usery/public_html`**

- Upload the zip file that contains the `/home/userx/public_html` files
- **Extract the zip file** You should now see all of the files in `/home/usery/public_html`
- Delete the zip file

- **Navigate to `/home/usery/elggdata`**
  - Upload the zip file that contains the `/home/userx/elggdata` files
  - **Extract the zip file** You should now see all of the files in /home/usery/elggdata
  - Delete the zip file

Program and data file restoration is complete

Restore the MySQL Database

Nota: Again, the assumption here is that you're restoring your Elgg installation to a second shared hosting provider. Each shared hosting provider prepends the account holder's name to the databases associated with that account. For example, the username for our primary host is `userx` so the host will prepend `userx_` to give us a database name of `userx_elgg`. When we restore to our second shared hosting provider we're doing so with a username of `usery` so our database name will be `usery_elgg`. The hosting providers don't allow you to modify this behavior. So the process here isn't as simple as just restoring the database from backup to the usery account. However, having said that, it's not terribly difficult either.

Edit the MySQL backup

Open the `.sql` file that you extracted from your backup in your favorite text editor. Comment out the following lines with a hash mark:

```
#CREATE DATABASE /*!32312 IF NOT EXISTS*/ `userx_elgg` /*!40100 DEFAULT CHARACTER SET␣
↪latin1 */;
#USE `userx_elgg`;
```

Save the file.

Create the new database

Perform the following:

- **Login to the CPanel application on the new host and click on the «MySQL Databases» icon**
  - Fill in the database name and click the «create» button. For our example we are going to stick with `elgg` which will give us a database name of `usery_elgg`
  - **You can associate an existing user with the new database, but to create a new user you will need to:**
    - Go to the «Add New User» section of the «MySQL Databases» page
    - Enter the username and password. For our example we're going to keep it simple and use `elgg` once again. This will give us a username of `usery_elgg`
  - **Associate the new user with the new database**

- Go to the «Add User To Database» section of the «MySQL Databases» page. Add the `usery_elgg` user to the `usery_elgg` database
- Select «All Privileges» and click the «Make Changes» button

Restore the production database

Now it's time to restore the MySQL backup file by importing it into our new database named «usery_elgg».

- **Login to the CPanel application on the new host and click on the «phpMyAdmin icon**
  - Choose the `usery_elgg` database in the left hand column
  - Click on the «import» tab at the top of the page
  - Browse to the `.sql` backup on your local computer and select it
  - Click the «Go» button on the bottom right side of the page

You should now see a message stating that the operation was successful

Bringing it all together

The restored elgg installation knows **nothing** about the new database name, database username, directory structure, etc. That's what we're going to address here.

Edit `/public_html/elgg-config/settings.php` on the new hosting provider to reflect the database information for the database that you just created.

```
// Database username
$CONFIG->dbuser = 'usery_elgg';

// Database password
$CONFIG->dbpass = 'dbpassword';

// Database name
$CONFIG->dbname = 'usery_elgg';

// Database server
// (For most configurations, you can leave this as 'localhost')
$CONFIG->dbhost = 'localhost';
```

Upload the `settings.php` file back to the new host - overwriting the existing file.

Open the phpMyAdmin tool on the new host from the CPanel. Select the `usery_elgg` database on the left and click the SQL tab on the top of the page. Run the following SQL queries against the `usery_elgg` database:

Change the installation path

```
UPDATE `elgg_datalists` SET `value` = "/home/usery/public_html/grid/" WHERE `name` =
↪"path";
```

Change the data directory

```
UPDATE `elgg_datalists` SET `value` = "/home/usery/elggdata/" WHERE `name` = "dataroot
↪";
```

Change the site URL (if this has changed)

```
UPDATE `elgg_sites_entity` SET `url` = "http://mynewdomain.com";
```

Change the filestore data directory

```
UPDATE elgg_metastrings set string = '/home/usery/elggdata/' WHERE id = (SELECT value_
↪id from elgg_metadata where name_id = (SELECT * FROM (SELECT id FROM elgg_
↪metastrings WHERE string = 'filestore::dir_root') as ms2) LIMIT 1);
```

Finalizing the new installation

Run the upgrade script by visiting the following URL: `http://mynewdomain.com/upgrade.php` . Do this step twice - back to back.

Update your DNS records so that your host name resolves to the new host's IP address if this is a permanent move.

2.6.4 Congratulations!

If you followed the steps outlined here you should now have a fully functional copy of your primary Elgg installation.

2.6.5 Related

FTP backup script

Here is an automated script for backing up an Elgg installation.

```
#!/usr/bin/perl -w

# FTP Backup

use Net::FTP;

# DELETE BACKUP AFTER FTP UPLOAD (0 = no, 1 = yes)
$delete_backup = 1;

# ENTER THE PATH TO THE DIRECTORY YOU WANT TO BACKUP, NO TRAILING SLASH
$directory_to_backup = '/home/userx/public_html';
$directory_to_backup2 = '/home/userx/elggdata';

# ENTER THE PATH TO THE DIRECTORY YOU WISH TO SAVE THE BACKUP FILE TO, NO TRAILING␣
↪SLASH
$backup_dest_dir = '/home/userx/sitebackups';

# BACKUP FILE NAME OPTIONS
($a,$d,$d,$day,$month,$yearoffset,$r,$u,$o) = localtime();
$year = 1900 + $yearoffset;
$site_backup_file = "$backup_dest_dir/site_backup-$day-$month-$year.tar.gz";
$full_backup_file = "$backup_dest_dir/full_site_backup-$day-$month-$year.tar.gz";

# MYSQL BACKUP PARAMETERS
$dbhost = 'localhost';
$dbuser = 'userx_elgg';
$dbpwd = 'dbpassword';
$mysql_backup_file_elgg = "$backup_dest_dir/mysql_elgg-$day-$month-$year.sql.gz";
```

(continué en la próxima página)

(proviene de la página anterior)

```
# ENTER DATABASE NAME
$database_names_elgg = 'userx_elgg';

# FTP PARAMETERS
$ftp_backup = 1;
$ftp_host = "FTP HOSTNAME/IP";
$ftp_user = "ftpuser";
$ftp_pwd = "ftppassword";
$ftp_dir = "/";

# SYSTEM COMMANDS
$cmd_mysqldump = '/usr/bin/mysqldump';
$cmd_gzip = '/usr/bin/gzip';

# CURRENT DATE / TIME
($a,$d,$d,$day,$month,$yearoffset,$r,$u,$o) = localtime();
$year = 1900 + $yearoffset;

# BACKUP FILES
$syscmd = "tar --exclude $backup_dest_dir" . "/* -czf $site_backup_file $directory_to_
↪backup $directory_to_backup2";

# elgg DATABASE BACKUP
system($syscmd);
$syscmd = "$cmd_mysqldump --host=$dbhost --user=$dbuser --password=$dbpwd --add-drop-
↪table --databases $database_names_elgg -c -l | $cmd_gzip > $mysql_backup_file_elgg";

system($syscmd);

# CREATING FULL SITE BACKUP FILE
$syscmd = "tar -czf $full_backup_file $mysql_backup_file_elgg $site_backup_file";
system($syscmd);

# DELETING SITE AND MYSQL BACKUP FILES
unlink($mysql_backup_file_elgg);
unlink($site_backup_file);

# UPLOADING FULL SITE BACKUP TO REMOTE FTP SERVER
if($ftp_backup == 1)
{
   my $ftp = Net::FTP->new($ftp_host, Debug => 0)
      or die "Cannot connect to server: $@";

   $ftp->login($ftp_user, $ftp_pwd)
      or die "Cannot login ", $ftp->message;

   $ftp->cwd($ftp_dir)
      or die "Can't CWD to remote FTP directory ", $ftp->message;

   $ftp->binary();

   $ftp->put($full_backup_file)
      or warn "Upload failed ", $ftp->message;

   $ftp->quit();
}
```

(continué en la próxima página)

(proviene de la página anterior)

```
# DELETING FULL SITE BACKUP
if($delete_backup = 1)
{
   unlink($full_backup_file);
}
```

Duplicate Installation

Contents

Introduction

Why Duplicate an Elgg Installation?

There are many reasons you may want to duplicate an Elgg installation: moving the site to another server, creating a test or development server, and creating functional backups are the most common. To create a successful duplicate of an Elgg site, 3 things need to be copied:

- Database
- Data from the data directory

- Code

Also at least 5 pieces of information must be changed from the copied installation:

- `elgg-config/settings.php` file which could also be in the pre 2.0 location `engine/settings.php`
- `.htaccess` file (Apache) or Nginx configuration depending on server used
- database entry for your site entity
- database entry for the installation path
- database entry for the data path

What Is Not Covered in This Tutorial

This tutorial expects a basic knowledge of Apache, MySQL, and Linux commands. As such, a few things will not be covered in this tutorial. These include:

- How to backup and restore MySQL databases
- How to configure Apache to work with Elgg
- How to transfer files to and from your production server

Before You Start

Before you start, make sure the Elgg installation you want to duplicate is fully functional. You will also need the following items:

- A backup of the live Elgg database
- A place to copy the live database
- **A server suitable for installing duplicate Elgg site** (This can be the same server as your production Elgg installation.)

Backups of the database can be obtained various ways, including phpMyAdmin, the MySQL official GUI, and the command line. Talk to your host for information on how to backup and restore databases or use Google to find information on this.

During this tutorial, we will make these assumptions about the production Elgg site:

- The URL is `http://www.myelgg.org/`
- The installation path is `/var/www/elgg/`
- The data directory is `/var/data/elgg/`
- The database host is `localhost`
- The database name is `production_elgg`
- The database user is `db_user`
- The database password is `db_password`
- The database prefix is `elgg`

At the end of the tutorial, our test Elgg installation details will be:

- The URL is `http://test.myelgg.org/`
- The installation path is `/var/www/elgg_test/`

- The data directory is `/var/data/elgg_test/`
- The database host is `localhost`
- The database name is `test_elgg`
- The database user is `db_user`
- The database password is `db_password`
- The database prefix is `elgg`

Copy Elgg Code to the Test Server

The very first step is to duplicate the production Elgg code. In our example, this is as simple as copying `/var/www/elgg/` to `/var/www/elgg_test/`.

```
cp -a /var/www/elgg/ /var/www/elgg_test/
```

Copy Data to the Test Server

In this example, this is as simple as copying `/var/data/elgg/` to `/var/data/elgg_test/`.

```
cp -a /var/data/elgg/ /var/data/elgg_test/
```

If you don't have shell access to your server and have to ftp the data, you may need to change ownership and permissions on the files.

Nota: You also need to delete the views cache on the test server after the copy process. This is a directory called `views_simplecache` in your data directory and the directory called `system_cache` .

Edit settings.php

The `elgg-config/settings.php` file contains the database configuration details. These need to be adjusted for your new test Elgg installation. In our example, we'll look in `/var/www/elgg_test/elgg-config/settings.php` and find the lines that look like this:

```
// Database username
$CONFIG->dbuser = 'db_user';

// Database password
$CONFIG->dbpass = 'db_password';

// Database name
$CONFIG->dbname = 'elgg_production';

// Database server
// (For most configurations, you can leave this as 'localhost')
$CONFIG->dbhost = 'localhost';

// Database table prefix
// If you're sharing a database with other applications, you will want to use this
```

(continué en la próxima página)

(proviene de la página anterior)

```
// to differentiate Elgg's tables.
$CONFIG->dbprefix = 'elgg';
```

We need to change these lines to match our new installation:

```
// Database username
$CONFIG->dbuser = 'db_user';

// Database password
$CONFIG->dbpass = 'db_password';

// Database name
$CONFIG->dbname = 'elgg_test';

// Database server
// (For most configurations, you can leave this as 'localhost')
$CONFIG->dbhost = 'localhost';

// Database table prefix
// If you're sharing a database with other applications, you will want to use this
// to differentiate Elgg's tables.
$CONFIG->dbprefix = 'elgg';
```

Nota: Notice the `$CONFIG->dbname` has changed to reflect our new database.

Copy Elgg Database

Now the database must be copied from `elgg_production` to `elgg_test`. See your favorite MySQL manager's documentation for how to make a duplicate database. You will generally export the current database tables to a file, create the new database, and then import the tables that you previously exported.

You have two options on updating the values in the database. You could change the values in the export file or you could import the file and change the values with database queries. One advantage of modifying the dump file is that you can also change links that people have created to content within your site. For example, if people have bookmarked pages using the bookmark plugin, the bookmarks will point to the old site unless your update their URLs.

Database Entries

We must now change 4 entries in the database. This is easily accomplished with 4 simple SQL commands:

Change the installation path

```
UPDATE `elgg_datalists` SET `value` = "/var/www/elgg_test/" WHERE `name` = "path";
```

Change the data directory

```
UPDATE `elgg_datalists` SET `value` = "/var/data/elgg_test/" WHERE `name` = "dataroot
↪";
```

Change the site URL

```
UPDATE `elgg_sites_entity` SET `url` = "http://test.myelgg.org/";
```

Change the filestore data directory

```
UPDATE elgg_metastrings SET string = '/var/data/elgg_test/'
WHERE id = (
   SELECT value_id
   FROM elgg_metadata
   WHERE name_id = (
      SELECT *
      FROM (
         SELECT id
         FROM elgg_metastrings
         WHERE string = 'filestore::dir_root'
      ) as ms2
   )
   LIMIT 1
);
```

Advertencia: Only change the first path above!!

Advertencia: If you have a plugin that uses custom filestores (contains an `ElggFile::setFilestore` method call or sets metadata with names like `filestore::*`), then query above may not be safe (it overwrites *all* filesystem `dir_root` locations). Please seek guidance via the Elgg community.

Check .htaccess

If you have made changes to .htaccess that modify any paths, make sure you update them in the test installation.

Update Webserver Config

For this example, you must edit the Apache config to enable a subdomain with a document root of `/var/www/elgg_test/`. If you plan to install into a subdirectory of your document root, this step is unnecessary.

If you're using Nginx, you need to update server config to match new paths based on `install/config/nginx.dist`.

Run upgrade.php

To regenerate cached data, make sure to run `http://test.myelgg.org/upgrade.php`

Tips

It is a good idea to keep a test server around to experiment with installing new mods and doing development work. If you automate restorations to the `elgg_test` database, changing the `$CONFIG` values and adding the follow lines to the end of the `elgg_test/elgg-config/settings.php` file will allow seamless re-writing of the MySQL database entries.

```
$con = mysql_connect($CONFIG->dbhost, $CONFIG->dbuser, $CONFIG->dbpass);
mysql_select_db($CONFIG->dbname, $con);

$sql = "UPDATE {$CONFIG->dbprefix}datalists
   SET value = '/var/www/test_elgg/'
   WHERE name = 'path'";
mysql_query($sql);
print mysql_error();

$sql = "UPDATE {$CONFIG->dbprefix}datalists
   SET value = '/var/data/test_elgg/'
   WHERE name = 'dataroot'";
mysql_query($sql);
print mysql_error();

$sql = "UPDATE {$CONFIG->dbprefix}sites_entity
   SET url = 'http://test.myelgg.org/'";
mysql_query($sql);

$sql = "UPDATE {$CONFIG->dbprefix}metastrings
  SET string = '/var/data/elgg_test/'
  WHERE id = (
     SELECT value_id
     FROM {$CONFIG->dbprefix}metadata
     WHERE name_id = (
        SELECT *
        FROM (
           SELECT id
           FROM {$CONFIG->dbprefix}metastrings
           WHERE string = 'filestore::dir_root'
        ) as ms2
     )
     LIMIT 1
  )";
mysql_query($sql);

print mysql_error();
```

Related

Ver también:

Backup and Restore

2.7 Getting Help

Having a problem with Elgg? The best way to get help is to ask at the Community Site. This site is community supported by a large group of volunteers. Here are a few tips to help you get the help you need.

Contents

2.7.1 Getting help

Don't be a Help Vampire

We were all newbies at one time, but we can all learn. Not showing that you are making attempts to learn on your own or do your own research is off putting for those helping. Also, very generic questions like «How do I build a forum?» are almost impossible to answer.

Search first

Be sure to search the documentation (this site), the Community Site, and Google before asking a question. New users to Elgg frequently have the same questions, so please search. People are less inclined to reply to a post that has been answered many other times or that can be answered easily by Googling.

Ask once

Posting the same questions in multiple places makes it hard to answer you. Ask your question in one place only. Duplicate questions may be moderated.

Include Elgg Version

Different versions of Elgg have different features (and different bugs). Including the version of Elgg that you are using will help those helping you.

Have a reasonable profile

Profiles that look like spam or have silly names will often be ignored. Joviality is fine, but people are more likely to help Michael than 1337elggHax0r.

Post in the appropriate forum

Check to make sure you're posting in the right forum. If you have a question about creating a plugin, don't post to the Elgg Feedback forum. If you need help installing Elgg, post to Technical Support instead of the Theming group.

Use a descriptive topic title

Good topic titles concisely describe your problem or question. Bad topic titles are vague, contain all capital letters, and excessive punctuation.

Good title: «White screen after upgrading to 1.7.4.»

Bad title: «URGENT!!!!! site broke ;-(losing money help!!!!!!!!!!»

Be detailed

Include as many details about your problem as possible. If you have a live site, include a link. Be forthcoming if community members might ask for more information. We can't help you if you won't give any details!

Keep it public

This is a public forum for the good of the Elgg project. Keep posts public. There's no reason for anyone to ask you to send a private message or email. Likewise, there's no reason to ask anyone to send a private email to you. Post in the public.

2.7.2 Guidelines

In addition to the site-wide Terms and Policies, following these guidelines keeps our community site useful and safe for everyone.

Content

All content must be safe for work: PG in the US and UK. If your Elgg site has adult content and you have been asked to post a link, please mark it NSFW (Not Safe For Work) so people know.

Excessive swearing in any language will not be tolerated.

Mood

Working with technical problems can be frustrating. Please keep the community site free of frustration. If you're feeling anxious, take a step away and do something else. Threatening or attacking community members, core developers, or plugin developers will not help solve your problem and will likely get you banned.

Advertising

Advertising is not allowed. Posts with any sort of advertising will be moderated.

Asking for money / Offering to pay

Don't ask for money on the community site. Likewise, don't offer to pay for answers. If you are looking for custom development, post to the Professional Services group. Posts asking for money or recommending a commercial plugin may be moderated.

Links

If you're having a problem with a live site, please provide a link to it.

That said, the community site is not a back linking service or SEO tool. Excessive linking will be moderated and your account may be banned.

Signatures

There's a reason Elgg doesn't have an option for signatures: they cause clutter and distract from the conversation. Users are discouraged from using signatures on the community site, and signatures with links or advertising will be removed.

Bumping, +1, me too

Don't do it. If your question hasn't been answered, see the top of this document for tips. These types of post add nothing to the conversation and may be moderated.

Posting Code

Long bits of code are confusing to read through in a forums context. Please use http://elgg.pastebin.com to post long bits of code and provide the Paste Bin link instead of directly posting the code.

2.7.3 Good Ideas

Not policies, but good ideas.

Say thanks

Did someone help you? Be sure to thank them! The community site is run by volunteers. No one has to help you with your problem. Be sure to show your appreciation!

Give back

Have a tip for Elgg? See someone with a similar problem you had? You've been there and can help them out, so give them a hand!

CAPÍTULO 3

Guías para desarrolladores

Personalice el comportamiento de Elgg mediante complementos.

3.1 Don't Modify Core

Advertencia: In general, you shouldn't modify non-config files that come with third-party software like Elgg.

The best way to customize the behavior of Elgg is to *install Elgg as a composer dependency* and use the root directory to store modifications specific to your application, and alter behavior through the rich Elgg plugin API.

If you'd like to share customizations between sites or even publish your changes as a reusable package for the community, create a *plugin* using the same plugin APIs and file structure.

3.1.1 It makes it hard to get help

When you don't share the same codebase as everyone else, it's impossible for others to know what is going on in your system and whether your changes are to blame. This can frustrate those who offer help because it can add considerable noise to the support process.

3.1.2 It makes upgrading tricky and potentially disastrous

You will certainly want or need to upgrade Elgg to take advantage of

- security patches
- new features
- new plugin APIs
- new stability improvements

- performance improvements

If you've modified core files, then you must be very careful when upgrading that your changes are not overwritten and that they are compatible with the new Elgg code. If your changes are lost or incompatible, then the upgrade may remove features you've added and even completely break your site.

This can also be a slippery slope. Lots of modifications can lead you to an upgrade process so complex that it's practically impossible. There are lots of sites stuck running old versions software due to taking this path.

3.1.3 It may break plugins

You may not realize until much later that your «quick fix» broke seemingly unrelated functionality that plugins depended on.

3.1.4 Summary

- **Resist the temptation** Editing existing files is quick and easy, but doing so heavily risks the maintainability, security, and stability of your site.
- When receiving advice, consider if the person telling you to modify core will be around to rescue you if you run into trouble later!
- **Apply these principle to software in general.** If you can avoid it, don't modify third party plugins either, for the same reasons: Plugin authors release new versions, too, and you will want those updates.

3.2 Complementos

Plugins must provide a start.php and manifest.xml file in the plugin root in order to be recognized by Elgg.

3.2.1 start.php

The start.php file bootstraps plugin by registering event listeners and plugin hooks.

3.2.2 elgg-plugin.php

This optional file is read by Elgg to configure various services, and must return an array if present. It should not be included by plugins and is not guaranteed to run at any particular time. Besides magic constants like `__DIR__`, its return value should not change.

Syntax

Here's a trivial example configuring view locations via the `views` key:

```
<?php

return [
        'views' => [
                'default' => [
                        'file/icon/' => __DIR__ . '/graphics/icons',
                ],
```

(continué en la próxima página)

(proviene de la página anterior)

```
        ],
];
```

3.2.3 activate.php, deactivate.php

The activate.php and deactivate.php files contain procedural code that will run upon plugin activation and deactivation. Use these files to perform one-time events such as registering a persistent admin notice, registering subtypes, or performing garbage collection when deactivated.

3.2.4 manifest.xml

Elgg plugins are required to have a `manifest.xml` file in the root of a plugin.

The `manifest.xml` file includes information about the plugin itself, requirements to run the plugin, and optional information including where to display the plugin in the admin area and what APIs the plugin provides.

Syntax

The manifest file is a standard XML file in UTF-8. Everything is a child of the `<plugin_manifest>` element.

```
<?xml version="1.0" encoding="UTF-8" ?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
```

The manifest syntax is as follows:

```
<name>value</name>
```

Many elements can contain children attributes:

```
<parent_name>
        <child_name>value</child_name>
        <child_name_2>value_2</child_name_2>
</parent_name>
```

Required Elements

All plugins are required to define the following elements in their manifest files:

- id - This has the name as the directory that the plugin uses.
- name - The display name of the plugin.
- author - The name of the author who wrote the plugin.
- version - The version of the plugin.
- description - A description of the what the plugin provides, its features, and other relevant information
- requires - Each plugin must specify the release of Elgg it was developed for. See the plugin Dependencies page for more information.

Available Elements

In addition to the require elements above, the follow elements are available to use:

- blurb - A short description of the plugin.
- category - The category of the plugin. It is recommended to follow the [[Plugin_Guidelines|plugin guidelines]] and use one of the defined categories. There can be multiple entries.
- conflicts - Specifies that the plugin conflicts with a certain system configuration.
- copyright - The plugin's copyright information.
- license - The plugin's license information.
- provides - Specifies that this plugin provides the same functionality as another Elgg plugin or a PHP extension.
- screenshot - Screenshots of the plugin. There can be multiple entries. See the advanced example for syntax.
- suggests - Parallels the requires system, but doesn't affect if the plugin can be enabled. Used to suggest other plugins that interact or build on the plugin.
- website - A link to the website for the plugin.

Ver también:

Plugin Dependencies

Ejemplo simple

This manifest file is the bare minimum a plugin must have.

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
        <name>Example Manifest</name>
        <author>Elgg</author>
        <version>1.0</version>
        <description>This is a simple example of a manifest file. In this example,␣
↪there are not screenshots, dependencies, or additional information about the plugin.
↪</description>

        <requires>
                <type>elgg_release</type>
                <version>1.9</version>
        </requires>
</plugin_manifest>
```

Advanced example

This example uses all of the available elements:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
        <name>Example Manifest</name>
        <author>Brett Profitt</author>
        <version>1.0</version>
        <blurb>This is an example manifest file.</blurb>
        <description>This is a simple example of a manifest file. In this example,␣
↪there are many options used, including screenshots, dependencies, and additional␣
↪information about the plugin.</description>
```

(continué en la próxima página)

(proviene de la página anterior)

```
        <website>http://www.elgg.org/</website>
        <copyright>(C) Brett Profitt 2014</copyright>
        <license>GNU Public License version 2</license>

        <category>3rd_party_integration</category>

        <requires>
                <type>elgg_release</type>
                <version>1.9.1</version>
        </requires>

        <!-- The path is relative to the plugin's root. -->
        <screenshot>
                <description>Elgg profile.</description>
                <path>screenshots/profile.png</path>
        </screenshot>

        <provides>
                <type>plugin</type>
                <name>example_plugin</name>
                <version>1.5</version>
        </provides>

        <suggests>
                <type>plugin</type>
                <name>twitter</name>
                <version>1.0</version>
        </suggests>
</plugin_manifest>
```

3.2.5 Related

Plugin skeleton

The following is the standard for plugin structure in Elgg as of Elgg 2.0.

Example Structure

The following is an example of a plugin with standard structure. For further explanation of this structure, see the details in the following sections. Your plugin may not need all the files listed

The following files for plugin `example` would go in `/mod/example/`

```
actions/
    example/
        action.php
        other_action.php
classes/
    VendorNamespace/
        ExampleClass.php
languages/
    en.php
vendors/
```

(continué en la próxima página)

(proviene de la página anterior)

```
    example_3rd_party_lib/
views/
    default/
        example/
          component.css
          component.js
          component.png
        forms/
            example/
                action.php
                other_action.php
        object/
            example.php
            example/
                context1.php
                context2.php
        plugins/
            example/
                settings.php
                usersettings.php
        resources/
            example/
                all.css
                all.js
                all.php
                owner.css
                owner.js
                owner.php
        widgets/
            example_widget/
                content.php
                edit.php
activate.php
deactivate.php
elgg-plugin.php
CHANGES.txt
COPYRIGHT.txt
INSTALL.txt
LICENSE.txt
manifest.xml
README.txt
start.php
```

Required Files

Plugins **must** provide a `start.php` and `manifest.xml` file in the plugin root in order to be recognized by Elgg.

Therefore the following is the minimally compliant structure:

```
mod/example/
    start.php
    manifest.xml
```

Actions

Plugins *should* place scripts for actions an `actions/` directory, and furthermore *should* use the name of the action to determine the location within that directory.

For example, the action `my/example/action` would go in `my_plugin/actions/my/example/action.php`. This makes it very obvious which script is associated with which action.

Similarly, the body of the form that submits to this action should be located in `forms/my/example/action.php`. Not only does this make the connection b/w action handler, form code, and action name obvious, but it allows you to use the new (as of Elgg 1.8) `elgg_view_form()` function easily.

Text Files

Plugins *may* provide various *.txt as additional documentation for the plugin. These files **must** be in Markdown syntax and will generate links on the plugin management sections.

README.txt *should* provide additional information about the plugin of an unspecified nature

COPYRIGHT.txt If included, **must** provide an explanation of the plugin's copyright, besides what is included in `manifest.xml`

LICENSE.txt If included, **must** provide the text of the license that the plugin is released under.

INSTALL.txt If included, **must** provide additional instructions for installing the plugin if the process is sufficiently complicated (e.g. if it requires installing third party libraries on the host machine, or requires acquiring an API key from a third party).

CHANGES.txt If included, **must** provide a list of changes for their plugin, grouped by version number, with the most recent version at the top.

Plugins *may* include additional *.txt files besides these, but no interface is given for reading them.

Pages

To render full pages, plugins should use **resource views** (which have names beginning with `resources/`). This allows other plugins to easily replace functionality via the view system.

Nota: The reason we encourage this structure is

- To form a logical relationship between urls and scripts, so that people examining the code can have an idea of what it does just by examining the structure.
- To clean up the root plugin directory, which historically has quickly gotten cluttered with the page handling scripts.

Classes

Elgg provides PSR-0 autoloading out of every active plugin's `classes/` directory.

You're encouraged to follow the PHP-FIG standards when writing your classes.

Nota: Files with a «.class.php» extension will **not** be recognized by Elgg.

Vendors

Included third-party libraries of any kind *should* be included in the `vendors/` folder in the plugin root. Though this folder has no special significance to the Elgg engine, this has historically been the location where Elgg core stores its third-party libraries, so we encourage the same format for the sake of consistency and familiarity.

Vistas

In order to override core views, a plugin's views can be placed in `views/`, or an `elgg-plugin.php` config file can be used for more detailed file/path mapping. See *Vistas*.

Javascript and CSS will live in the views system. See *JavaScript*.

activate.php and deactivate.php

The `activate.php` and `deactivate.php` files contain procedural code that will run respectively upon plugin activation or deactivation. Use these files to perform one-time events such as registering a persistent admin notice, registering subtypes, or performing garbage collection when deactivated.

Plugin Dependencies

In Elgg 1.8 a plugin dependencies system was introduced to prevent plugins from being used on incompatible systems.

Contents

- *Requires the Groups plugin is active*
- *Requires to be after the Profile plugin if Profile is active*
- *Conflicts with The Wire plugin*
- *Requires at least 256 MB memory in PHP*
- *Requires at least PHP version 5.3*
- *Suggest the TidyPics plugin is loaded*

Resumen

The dependencies system is controlled through a plugin's `manifest.xml` file. Plugin authors can specify that a plugin:

- Requires certain Elgg versions, Elgg plugins, PHP extensions, and PHP settings.
- Suggests certain Elgg versions, Elgg plugins, PHP extensions, and PHP settings.
- Conflicts with certain Elgg versions or Elgg plugins.
- Provides the equivalent of another Elgg plugin or PHP extension.

The dependency system uses the four verbs above (`requires`, `suggests`, `conflicts`, and `provides`) as parent elements to indicate what type of dependency is described by its children. All dependencies have a similar format with similar options:

```
<verb>
   <type>type</type>
   <noun>value</noun>
   <noun2>value2</noun2>
</verb>
```

Nota: `type` is always required

Verbs

With the exception of `provides`, all verbs use the same six types with differing effects, and the type options are the same among the verbs. `provides` only supports `plugin` and `php_extension`.

Requires

Using a `requires` dependency means that the plugin cannot be enabled unless the dependency is exactly met.

Mandatory requires: elgg_release

Every plugin must have at least one requires: the version of Elgg the plugin is developed for. This is specified by the Elgg API `release` (1.8). The default comparison >=, but you can specify your own by passing the `<comparison>` element.

Using elgg_release:

```
<requires>
   <type>elgg_release</type>
   <version>1.8</version>
</requires>
```

Suggests

`suggests` dependencies signify that the plugin author suggests a specific system configuration, but it is not required to use the plugin. The suggestions can also be another plugin itself which could interact, extend, or be extended by this plugin, but is not required for it to function.

Suggest another plugin:

```
<suggests>
   <type>plugin</type>
   <name>twitter_api</name>
   <version>1.0</version>
</suggests>
```

Suggest a certain PHP setting:

```
<suggests>
   <type>php_ini</type>
   <name>memory_limit</name>
   <value>64M</value>
   <comparison>ge</comparison>
</suggests>
```

Conflicts

`conflicts` dependencies mean the plugin cannot be used under a specific system configuration.

Conflict with any version of the profile plugin:

```
<conflicts>
   <type>plugin</type>
   <name>profile</name>
</conflicts>
```

Conflict with a specific release of Elgg:

```
<conflicts>
   <type>elgg_release</type>
   <version>1.8</version>
   <comparison>==</comparison>
</conflicts>
```

Provides

`provides` dependencies tell Elgg that this plugin is providing the functionality of another plugin or PHP extension. Unlike the other verbs, it only supports two types: `plugin` and `php_extension`.

The purpose of this is to provide interchangeable APIs implemented by different plugins. For example, the twitter_services plugin provides an API for other plugins to Tweet on behalf of the user via curl and Oauth. A plugin author could write a compatible plugin for servers without curl support that uses sockets streams and specify that it provides twitter_services. Any plugins that suggest or require twitter_services would then know they can work.

```
<provides>
   <type>plugin</type>
   <name>twitter_services</name>
   <version>1.8</version>
</provides>
```

Nota: All plugins provide themselves as their plugin id (directory name) at the version defined in the their manifest.

Types

Every dependency verb has a mandatory `<type>` element that must be one of the following six values:

1. **elgg_release** - The release version of Elgg (1.8)
2. **plugin** - An Elgg plugin
3. **priority** - A plugin load priority
4. **php_extension** - A PHP extension
5. **php_ini** - A PHP setting
6. **php_version** - A PHP version

Nota: `provides` only supports `plugin` and `php_extension` types.

Every type is defined with a dependency verb as the parent element. Additional option elements are at the same level as the type element:

```
<verb>
   <type>type</type>
   <option_1>value_1</option_1>
   <option_2>value_2</option_2>
</verb>
```

elgg_release

These concern the API and release versions of Elgg and requires the following option element:

- **version** - The API or release version

The following option element is supported, but not required:

- **comparison** - The comparison operator to use. Defaults to >= if not passed

plugin

Specifies an Elgg plugin by its ID (directory name). This requires the following option element:

- **name** - The ID of the plugin

The following option elements are supported, but not required:

- **version** - The version of the plugin
- **comparison** - The comparison operator to use. Defaults to >= if not passed

priority

This requires the plugin to be loaded before or after another plugin, if that plugin exists. `requires` should be used to require that a plugin exists. The following option elements are required:

- **plugin** - The plugin ID to base the load order on
- **priority** - The load order: "before" or "after"

php_extension

This checks PHP extensions. The follow option element is required:

- **name** - The name of the PHP extension

The following option elements are supported, but not required:

- **version** - The version of the extension
- **comparison** - The comparison operator to use. Defaults to ==

Nota: The format of extension versions varies greatly among PHP extensions and is sometimes not even set. This is generally worthless to check.

---

php_ini

This checks PHP settings. The following option elements are required:

- **name** - The name of the setting to check
- **value** - The value of the setting to compare against

The following options are supported, but not required:

- **comparison** - The comparison operator to use. Defaults to ==

php_version

This checks the PHP version. The following option elements are required:

- **version** - The PHP version

The following option element is supported, but not required:

- **comparison** - The comparison operator to use. Defaults to >= if not passed

Comparison Operators

Dependencies that check versions support passing a custom operator via the `<comparison>` element.

The follow are valid comparison operators:

- < or lt
- <= or le
- =, ==, or eq
- !=, <>, or ne
- > or gt
- >= or ge

If `<comparison>` is not passed, the follow are used as defaults, depending upon the dependency type:

- requires->elgg_release: >=
- requires->plugin: >=
- requires->php_extension: =
- requires->php_ini: =
- all conflicts: =

Nota: You must escape < and > to `&gt;` and `&lt;`. For comparisons that use these values, it is recommended you use the string equivalents instead!

Quick Examples

Requires Elgg 1.8.2 or higher

```
<requires>
   <type>elgg_release</type>
   <version>1.8.2</version>
</requires>
```

Requires the Groups plugin is active

```
<requires>
   <type>plugin</type>
   <name>groups</name>
</requires>
```

Requires to be after the Profile plugin if Profile is active

```
<requires>
   <type>priority</type>
   <priority>after</priority>
   <plugin>profile</plugin>
</requires>
```

Conflicts with The Wire plugin

```
<conflicts>
   <type>plugin</type>
   <name>thewire</name>
</conflicts>
```

Requires at least 256 MB memory in PHP

```
<requires>
   <type>php_ini</type>
   <name>memory_limit</name>
   <value>256M</value>
   <comparison>ge</comparison>
</requires>
```

Requires at least PHP version 5.3

```
<requires>
   <type>php_version</type>
   <version>5.3</version>
</requires>
```

Suggest the TidyPics plugin is loaded

```
<suggests>
   <type>plugin</type>
   <name>tidypics</name>
</suggests>
```

3.3 Plugin coding guidelines

In addition to the Elgg Coding Standards, these are guidelines for creating plugins. Core plugins are being updated to this format and all plugin authors should follow these guidelines in their own plugins.

Ver también:

Be sure to follow the *Plugin skeleton* for your plugin's layout.

Advertencia: *Don't Modify Core*

Contents

3.3.1 Use encaminamientos estándar con gestores de páginas

- Example: Bookmarks plugin
- **Page handlers should accept the following standard URLs:**

Purpose	URL
All	page_handler/all
User	page_handler/owner/<username>
User friends'	page_handler/friends/<username>
Single entity	page_handler/view/<guid>/<title>
Add	page_handler/add/<container_guid>
Edit	page_handler/edit/<guid>
Group list	page_handler/group/<guid>/owner

- Include page handler scripts from the page handler. Almost every page handler should have a page handler script. (Example: `bookmarks/all` => `mod/bookmarks/views/default/resources/bookmarks/all.php`)
- Pass arguments like entity guids to the resource view via `$vars` in `elgg_view_resource()`.
- Call `elgg_gatekeeper()` and `elgg_admin_gatekeeper()` in the page handler function if required.
- The group URL should use views like `resources/groups/*.php` to render pages.
- Los gestores de páginas no deberían contener HTML.
- If upgrading a 1.7 plugin, update the URLs throughout the plugin. (Don't forget to remove `/pg/`!)

3.3.2 Use gestores de páginas y scripts estandarizados

- Example: Bookmarks plugin
- Store page functionality in `mod/<plugin>/views/default/resources/<page_handler>/<page_name>.php`
- Use `elgg_view_resource('<page_handler>/<page_name>')` to render that.
- Use the content page layout in page handler scripts: `$content = elgg_view_layout('content', $options);`

- Page handler scripts should not contain HTML
- Llame a `elgg_push_breadcrumb()` en los scripts de gestión de páginas.
- No need to worry about setting the page owner if the URLs are in the standardized format
- For group content, check the `container_guid` by using `elgg_get_page_owner_entity()`

3.3.3 The object/<subtype> view

- Example: Bookmarks plugin
- Make sure there are views for `$vars['full_view'] == true` and `$vars['full_view'] == false`
- Check for the object in `$vars['entity']` . Use `elgg_instance_of()` to make sure it's the type entity you want. Return `true` to short circuit the view if the entity is missing or wrong.
- Use the new list body and list metadata views to help format. You should use almost no markup in these views.
- Update action structure - Example: Bookmarks plugin.
- Use espacios de nombre para ficheros y nombres de acciones. Por ejemplo, `mod/blog/actions/blog/save.php` → `action/blog/save`.
- **Use las siguientes direcciones URL de acciones:**

Purpose	URL
Add	action/plugin/save
Edit	action/plugin/save
Delete	action/plugin/delete

- Make the delete action accept `action/<handler>/delete?guid=<guid>` so the metadata entity menu has the correct URL by default
- If updating a 1.7 plugin, replace calls to functions deprecated in 1.7 because these will produce visible errors on every load in 1.8

3.3.4 Actions

Actions are transient states to perform an action such as updating the database or sending a notification to a user. Used correctly, actions provide a level of access control and prevent against CSRF attacks.

Actions require action (CSRF) tokens to be submitted via GET/POST, but these are added automatically by elgg_view_form() and by using the `is_action` argument of the `output/url` view.

Action best practices

Action files are included within Elgg's action system; like views, they are *not* regular scripts executable by users. Do not boot the Elgg core in your file and direct users to load it directly.

Because actions are time-sensitive they are not suitable for links in emails or other delayed notifications. An example of this would be invitations to join a group. The clean way to create an invitation link is to create a page handler for invitations and email that link to the user. It is then the page handler's responsibility to create the action links for a user to join or ignore the invitation request.

Consider that actions may be submitted via XHR requests, not just links or form submissions.

3.3.5 Directly calling a file

This is an easy one: **Don't do it**. With the exception of 3rd party application integration, there is not a reason to directly call a file in mods directory.

3.3.6 Recommended

These points are good ideas, but are not yet in the official guidelines. Following these suggestions will help to keep your plugin consistent with Elgg core.

- Update the widget views (see the blog or file widgets)
- Update the group profile "widget" using blog or file plugins as example
- **Update the forms**
    - Move form bodies to `/forms/<handler>/<action>` to use Evan's new `elgg_view_form()`
    - Use input views in form bodies rather than html
    - Add a function that prepares the form (see `mod/file/lib/file.php` for example)
    - Integrate sticky forms (see the file plugin's upload action and form prepare function)
- **Haz limpieza de CSS y HTML**
    - Should be able to remove almost all CSS (look for patterns that can be moved into core if you need CSS)
- Use hyphens rather than underscores in classes/ids
- Update the `manifest.xml` file to the 1.8 format. Use http://el.gg/manifest17to18 to automate this
- Do not use the `bundled` category with your plugins. That is for plugins distributed with Elgg
- **Update functions deprecated in 1.8.**
    - Many registration functions simply added an `elgg_` prefix for consistency
    - See `/engine/lib/deprecated-1.8.php` for the full list. You can also set the debug level to warning to get visual reminders of deprecated functions
- Don't use `register_shutdown_function` as you may not have access to certain Elgg parts anymore (eg database). Instead use the `shutdown system` event

3.4 Accessibility

This page aims to list and document accessibility rules and best practices, to help core and plugins developpers to make Elgg the most accessible social engine framework that everyone dreams of.

Nota: This is an ongoing work, please contribute on Github if you have some skills in this field!

3.4.1 Resources + references

- Official WCAG Accessibility Guidelines Overview
- Official WCAG Accessibility Guidelines

- Resources for planning and implementing for accessibility
- Practical tips from the W3C for improving accessibility
- Preliminary review of websites for accessibility
- Tools for checking the accessibility of websites
- List of practical techniques for implementing accessibility (It would be great if someone could go through this and filter out all the ones that are relevant to Elgg)

3.4.2 Tips for implementing accessibility

- All accessibility-related tickets reported to trac should be tagged with «a11y», short for «accessibility»
- Use core views such as `output/*`, and `input/*` to generate markup, since we can bake a11y concerns into these views
- All images should have a descriptive `alt` attribute. Spacer or purely decorative graphics should have blank `alt` attributes
- All `<a>` tags should have text or an accessible image inside. Otherwise screen readers will have to read the URL, which is a poor experience `<a>` tags should contain descriptive text, if possible, as opposed to generic text like «Click here»
- Markup should be valid
- Themes should not reset «outline» to nothing. `:focus` deserves a special visual treatment so that handicapped users can know where they are

3.4.3 Tips for testing accessibility

- Use the tools linked to from the resources section. Example report for community.elgg.org on June 16, 2012
- Try different font-size/zoom settings in your browser and make sure the theme remains usable
- Turn off css to make sure the sequential order of the page makes sense

3.4.4 Documentation objectives and principles

- Main accessibility rules
- collect and document best practices
- Provide code examples
- Keep the document simple and usable
- Make it usable for both beginner developpers and experts (from most common and easiest changes to elaborate techniques)

3.5 AJAX

The `elgg/Ajax` AMD module (introduced in Elgg 2.1) provides a set of methods for communicating with the server in a concise and uniform way, which allows plugins to collaborate on the request data, the server response, and the returned client-side data.

Client and server code written for the legacy API should not need modification.

Contents

3.5.1 Resumen

All the ajax methods perform the following:

1. Client-side, the `data` option (if given as an object) is filtered by the hook `ajax_request_data`.
2. The request is made to the server, either rendering a view or a form, calling an action, or loading a path.
3. The method returns a `jqXHR` object, which can be used as a Promise.
4. Server-echoed content is turned into a response object (`Elgg\Services\AjaxResponse`) containing a string (or a JSON-parsed value).
5. The response object is filtered by the hook `ajax_response`.
6. The response object is used to create the HTTP response.
7. Client-side, the response data is filtered by the hook `ajax_response_data`.
8. The `jqXHR` promise is resolved and any `success` callbacks are called.

More notes:

- All hooks have a type depending on the method and first argument. See below.
- By default the `elgg/spinner` module is automatically used during requests.
- User messages generated by `system_message()` and `register_error()` are collected and displayed on the client.
- Elgg gives you a default error handler that shows a generic message if output fails.
- PHP exceptions or denied resource return HTTP error codes, resulting in use of the client-side error handler.
- The default HTTP method is `POST` for actions, otherwise `GET`. You can set it via `options.method`.

- If a non-empty `options.data` is given, the default method is always `POST`.
- For client caching, set `options.method` to `"GET"` and `options.data.elgg_response_ttl` to the max-age you want in seconds.
- To save system messages for the next page load, set `options.data.elgg_fetch_messages = 0`. You may want to do this if you intent to redirect the user based on the response.
- To stop client-side API from requiring AMD modules required server-side with `elgg_require_js()`, set `options.data.elgg_fetch_deps = 0`.
- All methods accept a query string in the first argument. This is passed on to the fetch URL, but does not appear in the hook types.

Performing actions

Consider this action:

```
// in myplugin/actions/do_math.php

elgg_ajax_gatekeeper();

$arg1 = (int)get_input('arg1');
$arg2 = (int)get_input('arg2');

// will be rendered client-side
system_message('We did it!');

echo json_encode([
    'sum' => $arg1 + $arg2,
    'product' => $arg1 * $arg2,
]);
```

To execute it, use `ajax.action('<action_name>', options)`:

```
var Ajax = require('elgg/Ajax');
var ajax = new Ajax();

ajax.action('do_math', {
    data: {
        arg1: 1,
        arg2: 2
    },
}).done(function (output, statusText, jqXHR) {
    if (jqXHR.AjaxData.status == -1) {
        return;
    }

    alert(output.sum);
    alert(output.product);
});
```

Notes for actions:

- **All hooks have type `action:<action_name>`. So in this case, three hooks will be triggered:**
    - client-side `"ajax_request_data", "action:do_math"` to filter the request data (before it's sent)

    - server-side `"ajax_response", "action:do_math"` to filter the response (after the action runs)
    - client-side `"ajax_response_data", "action:do_math"` to filter the response data (before the calling code receives it)
- CSRF tokens are added to the request data.
- The default method is `POST`.
- An absolute action URL can be given in place of the action name.
- Using `forward()` in an action simply sends the response. The URL given in not returned to the client.

Nota: When setting `data`, use `ajax.objectify($form)` instead of `$form.serialize()`. Doing so allows the `ajax_request_data` plugin hook to fire and other plugins to alter/piggyback on the request.

Fetching data

Consider this PHP script that runs at `http://example.org/myplugin_time`.

```
// in myplugin/start.php
elgg_register_page_handler('myplugin_time', 'myplugin_get_time');

function myplugin_get_time() {
    elgg_ajax_gatekeeper();

    echo json_encode([
        'rfc2822' => date(DATE_RFC2822),
        'day' => date('l'),
    ]);

    return true;
}
```

To fetch its output, use `ajax.path('<url_path>', options)`.

```
var Ajax = require('elgg/Ajax');
var ajax = new Ajax();

ajax.path('myplugin_time').done(function (output, statusText, jqXHR) {
    if (jqXHR.AjaxData.status == -1) {
        return;
    }

    alert(output.rfc2822);
    alert(output.day);
});
```

Notes for paths:

- The 3 hooks (see Actions above) will have type `path:<url_path>`. In this case, «path:myplugin_time».
- If the page handler echoes a regular web page, `output` will be a string containing the HTML.
- An absolute URL can be given in place of the path name.

Fetching views

Consider this view:

```
// in myplugin/views/default/myplugin/get_link.php

if (empty($vars['entity']) || !$vars['entity'] instanceof ElggObject) {
    return;
}

$object = $vars['entity'];
/* @var ElggObject $object */

echo elgg_view('output/url', [
    'text' => $object->getDisplayName(),
    'href' => $object->getUrl(),
    'is_trusted' => true,
]);
```

Since it's a PHP file, we must register it for Ajax first:

```
// in myplugin_init()
elgg_register_ajax_view('myplugin/get_link');
```

To fetch the view, use `ajax.view('<view_name>', options)`:

```
var Ajax = require('elgg/Ajax');
var ajax = new Ajax();

ajax.view('myplugin/get_link', {
    data: {
        guid: 123 // querystring
    },
}).done(function (output, statusText, jqXHR) {
    if (jqXHR.AjaxData.status == -1) {
        return;
    }

    $('.myplugin-link').html(output);
});
```

Notes for views:

- The 3 hooks (see Actions above) will have type `view:<view_name>`. In this case, «view:myplugin/get_link».
- `output` will be a string with the rendered view.
- The request data are injected into `$vars` in the view.
- If the request data contains `guid`, the system sets `$vars['entity']` to the corresponding entity or `false` if it can't be loaded.

Advertencia: In ajax views and forms, note that `$vars` can be populated by client input. The data is filtered like `get_input()`, but may not be the type you're expecting or may have unexpected keys.

Fetching forms

Consider we have a form view. We register it for Ajax:

```
// in myplugin_init()
elgg_register_ajax_view('forms/myplugin/add');
```

To fetch this using `ajax.form('<action_name>', options)`.

```
var Ajax = require('elgg/Ajax');
var ajax = new Ajax();

ajax.form('myplugin/add').done(function (output, statusText, jqXHR) {
    if (jqXHR.AjaxData.status == -1) {
        return;
    }

    $('.myplugin-form-container').html(output);
});
```

Notes for forms:

- The 3 hooks (see Actions above) will have type `form:<action_name>`. In this case, «form:myplugin/add».
- `output` will be a string with the rendered view.
- The request data are injected into `$vars` in your form view.
- If the request data contains `guid`, the system sets `$vars['entity']` to the corresponding entity or `false` if it can't be loaded.

Nota: Only the request data are passed to the requested form view (i.e. as a third parameter accepted by `elgg_view_form()`). If you need to pass attributes or parameters of the form element rendered by the `input/form` view (i.e. normally passed as a second parameter to `elgg_view_form()`), use the server-side hook `view_vars, input/form`.

Advertencia: In ajax views and forms, note that `$vars` can be populated by client input. The data is filtered like `get_input()`, but may not be the type you're expecting or may have unexpected keys.

Piggybacking on an Ajax request

The client-side `ajax_request_data` hook can be used to append or filter data being sent by an `elgg/Ajax` request.

Let's say when the view `foo` is fetched, we want to also send the server some data:

```
// in your boot module
var Ajax = require('elgg/Ajax');
var elgg = require('elgg');

var ajax = new Ajax();

elgg.register_hook_handler(Ajax.REQUEST_DATA_HOOK, 'view:foo', function (name, type,␣
↪params, data) {
```

(continué en la próxima página)

(proviene de la página anterior)

```
    // send some data back
    data.bar = 1;
    return data;
});
```

This data can be read server-side via `get_input('bar');`.

Nota: If data was given as a string (e.g. `$form.serialize()`), the request hooks are not triggered.

Piggybacking on an Ajax response

The server-side `ajax_response` hook can be used to append or filter response data (or metadata).

Let's say when the view `foo` is fetched, we want to also send the client some additional data:

```
use Elgg\Services\AjaxResponse;

function myplugin_append_ajax($hook, $type, AjaxResponse $response, $params) {

    // alter the value being returned
    $response->getData()->value .= " hello";

    // send some metadata back. Only client-side "ajax_response" hooks can see this!
    $response->getData()->myplugin_alert = 'Listen to me!';

    return $response;
}

// in myplugin_init()
elgg_register_plugin_hook_handler(AjaxResponse::RESPONSE_HOOK, 'view:foo', 'myplugin_
↪append_ajax');
```

To capture the metadata send back to the client, we use the client-side `ajax_response` hook:

```
// in your boot module
var Ajax = require('elgg/Ajax');
var elgg = require('elgg');

elgg.register_hook_handler(Ajax.RESPONSE_DATA_HOOK, 'view:foo', function (name, type,␣
↪params, data) {

    // the return value is data.value

    // the rest is metadata

    alert(data.myplugin_alert);

    return data;
});
```

Nota: Only `data.value` is returned to the `success` function or available via the *Deferred* interface.

Nota: Elgg uses these same hooks to deliver system messages over `elgg/Ajax` responses.

Handling errors

Responses basically fall into three categories:

1. HTTP success (200) with status `0`. No `register_error()` calls were made on the server.
2. HTTP success (200) with status `-1`. `register_error()` was called.
3. HTTP error (4xx/5xx). E.g. calling an action with stale tokens, or a server exception. In this case the `done` and `success` callbacks are not called.

You may need only worry about the 2nd case. We can do this by looking at `jqXHR.AjaxData.status`:

```
ajax.action('entity/delete?guid=123').done(function (value, statusText, jqXHR) {
    if (jqXHR.AjaxData.status == -1) {
        // a server error was already displayed
        return;
    }

    // remove element from the page
});
```

Requiring AMD modules

Each response from an Ajax service will contain a list of AMD modules required server side with *elgg_require_js()*. When response data is unwrapped, these modules will be loaded asynchronously - plugins should not expect these modules to be loaded in their *$.done()* and *$.then()* handlers and must use *require()* for any modules they depend on. Additionally AMD modules should not expect the DOM to have been altered by an Ajax request when they are loaded - DOM events should be delegated and manipulations on DOM elements should be delayed until all Ajax requests have been resolved.

3.5.2 Legacy elgg.ajax APIs

Elgg 1.8 introduced `elgg.action`, `elgg.get`, `elgg.getJSON`, and other methods which behave less consistently both client-side and server-side.

Legacy elgg.action

Differences:

- you must manually pull the `output` from the returned wrapper
- the `success` handler will fire even if the action is prevented
- the `success` handler will receive a wrapper object. You must look for `wrapper.output`
- no ajax hooks

```
elgg.action('do_math', {
    data: {
        arg1: 1,
        arg2: 2
    },
    success: function (wrapper) {
        if (wrapper.output) {
            alert(wrapper.output.sum);
            alert(wrapper.output.product);
        } else {
            // the system prevented the action from running, but we really don't
            // know why
            elgg.ajax.handleAjaxError();
        }
    }
});
```

elgg.action notes

- It's best to echo a non-empty string, as this is easy to validate in the `success` function. If the action was not allowed to run for some reason, `wrapper.output` will be an empty string.
- You may want to use the *elgg/spinner* module.
- Elgg does not use `wrapper.status` for anything, but a call to `register_error()` causes it to be set to `-1`.
- If the action echoes a non-JSON string, `wrapper.output` will contain that string.
- `elgg.action` is based on `jQuery.ajax` and returns a `jqXHR` object (like a Promise), if you should want to use it.
- After the PHP action completes, other plugins can alter the wrapper via the plugin hook `'output', 'ajax'`, which filters the wrapper as an array (not a JSON string).
- A `forward()` call forces the action to be processed and output immediately, with the `wrapper.forward_url` value set to the normalized location given.
- To make sure Ajax actions can only be executed via XHR, use `elgg_ajax_gatekeeper()`.

elgg.action JSON response wrapper

```
{
    current_url: {String} "http://example.org/action/example/math", // not very useful
    forward_url: {String} "http://example.org/foo", ...if forward('foo') was called
    output: {String|Object} from echo in action
    status: {Number} 0 = success. -1 = an error was registered.
    system_messages: {Object}
}
```

Advertencia: It's probably best to rely only on the `output` key, and validate it in case the PHP action could not run for some reason, e.g. the user was logged out or a CSRF attack did not provide tokens.

Advertencia: If `forward()` is used in response to a legacy ajax request (e.g. `elgg.ajax`), Elgg will *always* respond with this wrapper, **even if not in an action**.

Legacy view fetching

A plugin can use a view script to handle XHR `GET` requests. Here's a simple example of a view that returns a link to an object given by its GUID:

```
// in myplugin_init()
elgg_register_ajax_view('myplugin/get_link');
```

```
// in myplugin/views/default/myplugin/get_link.php

if (empty($vars['entity']) || !$vars['entity'] instanceof ElggObject) {
    return;
}

$object = $vars['entity'];
/* @var ElggObject $object */

echo elgg_view('output/url', [
    'text' => $object->getDisplayName(),
    'href' => $object->getUrl(),
    'is_trusted' => true,
]);
```

```
elgg.get('ajax/view/myplugin/get_link', {
    data: {
        guid: 123 // querystring
    },
    success: function (output) {
        $('.myplugin-link').html(output);
    }
});
```

The Ajax view system works significantly differently than the action system.

- There are no access controls based on session status.
- Non-XHR requests are automatically rejected.
- GET vars are injected into `$vars` in the view.
- If the request contains `$_GET['guid']`, the system sets `$vars['entity']` to the corresponding entity or `false` if it can't be loaded.
- There's no «wrapper» object placed around the view output.
- System messages/errors shouldn't be used, as they don't display until the user loads another page.
- Depending on the view's suffix (.js, .html, .css, etc.), a corresponding Content-Type header is added.

Advertencia:

In ajax views and forms, note that `$vars` can be populated by client input. The data is filtered like `get_input()`, but may not be the type you're expecting or may have unexpected keys.

Returning JSON from a view

If the view outputs encoded JSON, you must use `elgg.getJSON` to fetch it (or use some other method to set jQuery's ajax option `dataType` to `json`). Your `success` function will be passed the decoded Object.

Here's an example of fetching a view that returns a JSON-encoded array of times:

```
elgg.getJSON('ajax/view/myplugin/get_times', {
    success: function (data) {
        alert('The time is ' + data.friendly_time);
    }
});
```

Legacy form fetching

If you register a form view (name starting with `forms/`), you can fetch it pre-rendered with `elgg_view_form()`. Simply use `ajax/form/<action>` (instead of `ajax/view/<view_name>`):

```
// in myplugin_init()
elgg_register_ajax_view('forms/myplugin/add');
```

```
elgg.get('ajax/form/myplugin/add', {
    success: function (output) {
        $('.myplugin-form-container').html(output);
    }
});
```

Only the request data are passed to the requested form view (i.e. as a third parameter accepted by `elgg_view_form()`). If you need to pass attributes or parameters of the form element rendered by the `input/form` view (i.e. normally passed as a second parameter to `elgg_view_form()`), use the server-side hook `view_vars, input/form`.

Advertencia:

In ajax views and forms, note that `$vars` can be populated by client input. The data is filtered like `get_input()`, but may not be the type you're expecting or may have unexpected keys.

Legacy helper functions

These functions extend jQuery's native Ajax features.

`elgg.get()` is a wrapper for jQuery's `$.ajax()`, but forces `GET` and does URL normalization.

```
// normalizes the url to the current <site_url>/activity
elgg.get('/activity', {
    success: function(resultText, success, xhr) {
        console.log(resultText);
    }
});
```

`elgg.post()` is a wrapper for jQuery's `$.ajax()`, but forces `POST` and does URL normalization.

3.6 Authentication

Elgg provides everything needed to authenticate users via username/email and password out of the box, including:

- remember-me cookies for persistent login
- password reset logic
- secure storage of passwords
- logout
- UIs for accomplishing all of the above

All that's left for you to do as a developer is to use the built-in authentication functions to secure your pages and actions.

3.6.1 Working with the logged in user

Check whether the current user is logged in with `elgg_is_logged_in()`:

```
if (elgg_is_logged_in()) {
  // do something just for logged-in users
}
```

Check if the current user is an admin with `elgg_is_admin_logged_in()`:

```
if (elgg_is_admin_logged_in()) {
  // do something just for admins
}
```

Get the currently logged in user with `elgg_get_logged_in_user_entity()`:

```
$user = elgg_get_logged_in_user_entity();
```

The returned object is an `ElggUser` so you can use all the methods and properties of that class to access information about the user. If the user is not logged in, this will return `null`, so be sure to check for that first.

3.6.2 Gatekeepers

Gatekeeper functions allow you to manage how code gets executed by applying access control rules.

Forward a user to the front page if they are not logged in with `elgg_gatekeeper()`:

```
elgg_gatekeeper();

echo "Information for logged-in users only";
```

Nota: In Elgg 1.8 and below this function was called `gatekeeper()`

Forward a user to the front page unless they are an admin with `elgg_admin_gatekeeper()`:

```
elgg_admin_gatekeeper();

echo "Information for admins only";
```

Nota: In Elgg 1.8 and below this function was called `admin_gatekeeper()`

Prevent CSRF attacks with `action_gatekeeper()`.

```
action_gatekeeper();

// Mutate some state in the database on behalf of the logged in user...
```

This function should be used in *Formularios y acciones* prior to Elgg 1.8.

Nota: As of Elgg version 1.8 this function is called for all registered actions. There is no longer a need to call this function in your own actions. If you wish to protect other pages with action tokens then you can call this function.

3.6.3 Pluggable Authentication Modules

Elgg has support for pluggable authentication modules (PAM), which enables you to write your own authentication handlers. Whenever a request needs to get authenticated the system will call `elgg_authenticate()` which probes the registered PAM handlers until one returns success.

The preferred approach is to create a separate Elgg plugin which will have one simple task: to process an authentication request. This involves setting up an authentication handler in the plugin's *start.php* file, and to register it with the PAM module so it will get processed whenever the system needs to authenticate a request.

The authentication handler is a function and takes a single parameter. Registering the handler is being done by `register_pam_handler()` which takes the name of the authentication handler, the importance and the policy as parameters. It is advised to register the handler in the plugin's init function, for example:

```
function your_plugin_init() {
   // Register the authentication handler
   register_pam_handler('your_plugin_auth_handler');
}

function your_plugin_auth_handler($credentials) {
   // do things ...
}

// Add the plugin's init function to the system's init event
elgg_register_elgg_event_handler('init', 'system', 'your_plugin_init');
```

3.6.4 Importance

By default an authentication module is registered with an importance of **sufficient**.

In a list of authentication modules; if any one marked *sufficient* returns `true`, `pam_authenticate()` will also return `true`. The exception to this is when an authentication module is registered with an importance of **required**. All required modules must return `true` for `pam_authenticate()` to return `true`, regardless of whether all sufficient modules return `true`.

3.6.5 Passed credentials

The format of the credentials passed to the handler can vary, depending on the originating request. For example, a regular login via the login form will create a named array, with the keys `username` and `password`. If a request was made for example via XML-RPC then the credentials will be set in the HTTP header, so in this case nothing will get passed to the authentication handler and the handler will need to perform steps on its own to authenticate the request.

3.6.6 Return value

The authentication handle should return a `boolean`, indicating if the request could be authenticated or not. One caveat is that in case of a regular user login where credentials are available as username and password the user will get logged in. In case of the XML-RPC example the authentication handler will need to perform this step itself since the rest of the system will not have any idea of either possible formats of credentials passed nor its contents. Logging in a user is quite simple and is being done by `login()`, which expects an `ElggUser` object.

3.7 Context

Within the Elgg framework, context can be used to by your plugin's functions to determine if they should run or not. You will be registering callbacks to be executed when particular *events are triggered*. Sometimes the events are generic and you only want to run your callback when your plugin caused the event to be triggered. In that case, you can use the page's context.

You can explicitly set the context with `set_context()`. The context is a string and typically you set it to the name of your plugin. You can retrieve the context with the function `get_context()`. It's however better to use `elgg_push_context($string)` to add a context to the stack. You can check if the context you want in in the current stack by calling `elgg_in_context($context)`. Don't forget to pop (with `elgg_pop_context()`) the context after you push one and don't need it anymore.

If you don't set it, Elgg tries to guess the context. If the page was called through the page handler, the context is set to the name of the handler which was set in `elgg_register_page_handler()`. If the page wasn't called through the page handler, it uses the name of your plugin directory. If it cannot determine that, it returns main as the default context.

Sometimes a view will return different HTML depending on the context. A plugin can take advantage of that by setting the context before calling `elgg_view()` on the view and then setting the context back. This is frequently done with the search context.

3.8 Cron

If you setup cron correctly as described in *Cron* special hooks will be triggered so you can register for these hooks from your own code.

The example below registers a function for the daily cron.

```
function my_plugin_init() {
    elgg_register_plugin_hook_handler('cron', 'daily', 'my_plugin_cron_handler');
}
```

If timing is important in your cron hook be advised that the functions are executed in order of registration. This could mean that your function may start (a lot) later then you may have expected. However the parameters provided in the hook contain the original starting time of the cron, so you can always use that information.

```
function my_plugin_cron_handler($hook, $period, $return, $params) {
    $start_time = elgg_extract('time', $params);
}
```

Ver también:

Eventos y ganchos de complementos has more information about hooks

3.9 Base de datos

Persiste contenido y opciones de usuarios mediante la API de almacenamiento genérico de Elgg.

Contents

3.9.1 Entidades

Crear un objeto

Para crear un objeto desde código, tiene que crear una instancia de `ElggObject`. Definir sus datos es simplemente cuestión de añadir variables o propiedades a la instancia. Las propiedades de serie son:

- **‘‘guid‘‘:** El identificador único de la entidad, definido de manera automática.
- **‘‘owner_guid‘‘:** El identificador único del usuario propietario del objeto.
- **‘‘site_guid‘‘:** El identificador único del sitio al que pertenece el objeto. Éste se define de manera automática al crear una instancia de `ElggObject`.
- **‘‘subtype‘‘:** Una cadena de texto arbitraria y sin espacios que define el tipo de objeto del que se trata, como `blog`.
- **‘‘access_id‘‘:** Un número entero que representa el nivel de acceso del objeto.

- **‘‘title‘‘:** El título del objeto.
- **‘‘description‘‘:** La descripción del objeto.

El subtipo del objeto es una propiedad especial. Se trata de una cadena de texto arbitraria que describe qué es el objeto. Por ejemplo, si estuviese escribiendo un complemento para blogs, su subtipo podría ser *blog*. Lo ideal es que la palabra sea única, de forma que otros complementos no la usen también de manera accidental. Para el propósito de esta documentación, partamos de la idea de crear un foro muy simple, para el que usaremos el subtipo *forum* («foro» en inglés):

```
$object = new ElggObject();
$object->subtype = "forum";
$object->access_id = 2;
$object->save();
```

`access_id` es otra propiedad importante. Se no le da usted valor a esta propiedad, el objeto será privado, y sólo el usuario creador del objeto podrá verlo. Elgg define constantes para valores especiales de `access_id`:

- **ACCESS_PRIVATE:** Sólo el usuario propietario del objeto puede verlo.
- **ACCESS_FRIENDS:** Sólo el usuario propietario del objeto y sus contactos pueden verlo.
- **ACCESS_LOGGED_IN:** Cualquier usuario registrado puede verlo.
- **ACCESS_PUBLIC:** Cualquier persona, con o sin cuenta en el sitio, puede verlo.

Al guardar el objeto, se le dará automáticamente un valor a su propiedad `guid` si la operación de guardado se completa correctamente. Si cambia más propiedades de serie, puede llamar al método `save()` del objeto de nuevo, y la base de datos se actualizará de acorde a sus cambios.

Puede definir metadatos en un objeto como haría con cualquier propiedad de serie. Digamos que queremos definir la edición (SKU) de un producto:

```
$object->SKU = 62784;
```

Si asigna un vector, todos los valores se definirán para ese metadato. Por ejemplo, así es como definiría etiquetas:

Los metadatos no se pueden persistir a la base de datos hasta que la entidad se ha guardado, pero por comodidad, `ElggEntity` puede cachearlos de manera interna y guardarlos al guardar la entidad.

Cargar un objeto

Por identificador

```
$entity = get_entity($guid);
if (!$entity) {
    // The entity does not exist or you're not allowed to access it.
}
```

¿Pero qué pasa si usted desconoce el identificador único? Pues existen varias opciones.

Por usuario, subtipo o sitio

Si conoce el identificador del usuario cuyos objetos está buscando, o el subtipo o el sitio de esos objetos, dispone de varias opciones para obtener esos objetos. La más fácil probablemente sea la de llamar a la función procedural `elgg_get_entities`:

```
$entities = elgg_get_entities(array(
    'type' => $entity_type,
    'subtype' => $subtype,
    'owner_guid' => $owner_guid,
));
```

Esto devolverá un vector de instancias de `ElggEntity` por el que usted puede iterar. `elgg_get_entities` usa divide los resultados en grupos de manera predeterminada, con un límite de 10 resultados por grupo, y empezando desde 0.

Puede omitir `owner_guid` para obtener todos los objetos u omitir el subtipo o el tipo para obtener objetos de todos los tipos o subtipos.

Si ya tiene una instancia de `ElggUser` —que puede obtener, por ejemplo, mediante `elgg_get_logged_in_user_entity` el objeto del usuario actual— puede usar:

```
$objects = $user->getObjects($subtype, $limit, $offset)
```

¿Pero qué hay de obtener objetos que tienen un metadato concreto?

Por metadatos

La función `elgg_get_entities_from_metadata` permite obtener entidades por metadatos de varias maneras.

By annotation

The function `elgg_get_entities_from_annotations` allows fetching entities with metadata in a variety of ways.

Nota: As of Elgg 1.10 the default behaviour of *elgg_get_entities_from_annotations* was brought inline with the rest of the *elgg_get_entities** functions.

Pre Elgg 1.10 the sorting of the entities was based on the latest addition of an annotation (in $options your could add *$options["order_by"] = "maxtime ASC"* or *$options["order_by"] = "maxtime DESC"*. As of Elgg 1.10 this was changed to the creation time of the entity, just like the rest of the *elgg_get_entities** functions. To get the old behaviour back add the following to your *$options*:

```
$options['selects'] = array('MAX(n_table.time_created) AS maxtime');
$options['group_by'] = 'n_table.entity_guid';
$options['order_by'] = 'maxtime ASC'

or

$options['order_by'] = 'maxtime DESC'
```

Mostrar entidades

In order for entities to be displayed in listing functions you need to provide a view for the entity in the views system.

Para mostrar una entidad, cree una vista llamada «TipoDeEntidad/subtipo» donde «TipoDeEntidad» es uno de los siguientes:

`object`, rara entidades derivadas de `ElggObject`; `user`, para entidades derivadas de `ElggUser`; `site`, para entidades derivadas de `ElggSite`; o `group`, para entidades derivadas de `ElggGroup`.

Ya se crea una vista predeterminada para todas las entidades, llamada «TipoDeEntidad/default».

Iconos de entidades

Entity icons can be saved from uploaded files, existing local files, or existing ElggFile objects. These methods save all sizes of icons defined in the system.

```
$object = new ElggObject();
$object->title = 'Example entity';
$object->description = 'An example object with an icon.';

// from an uploaded file
$object->saveIconFromUploadedFile('file_upload_input');

// from a local file
$object->saveIconFromLocalFile('/var/data/generic_icon.png');

// from a saved ElggFile object
$file = get_entity(123);
if ($file instanceof ElggFile) {
        $object->saveIconFromElggFile($file);
}

$object->save();
```

The following sizes exist by default:

- `master` - 550px at longer edge (not upscaled)
- `large` - 200px at longer edge (not upscaled)
- `medium` - 100px square
- `small` - 40px square
- `tiny` - 25px square
- `topbar` - 16px square

Use `elgg_get_icon_sizes()` to get all possible icon sizes for a specific entity type and subtype. The function triggers the `entity:icon:sizes` *hook*.

To check if an icon is set, use `$object->hasIcon($size)`.

You can retrieve the URL of the generated icon with‘‘ElggEntity::getIconURL($params)‘‘ method. This method accepts a `$params` argument as an array that specifies the size, type, and provide additional context for the hook to determine the icon to serve. The method triggers the `entity:icon:url` *hook*.

Use `elgg_view_entity_icon($entity, $size, $vars)` to render an icon. This will scan the following locations for a view and include the first match to .

1. views/$viewtype/icon/$type/$subtype.php
2. views/$viewtype/icon/$type/default.php
3. views/$viewtype/icon/default.php

Donde:

$viewtype Type of view, e.g. `'default'` or `'json'`.

$type Type of entity, e.g. `'group'` or `'user'`.

$subtype Entity subtype, e.g. `'blog'` or `'page'`.

Icon methods support passing an icon type if an entity has more than one icon. For example, a user might have an avatar and a cover photo icon. You would pass `'cover_photo'` as the icon type:

```
$object->saveIconFromUploadedFile('uploaded_photo', 'cover_photo');

$object->getIconUrl([
        'size' => 'medium',
        'type' => 'cover_photo'
]);
```

Note that custom icon types (e.g. cover photos) do not have preset sizes and coordinates. Use `entity:<icon_type>:url` *hook* to configure them.

By default icons will be stored in `/icons/<icon_type>/<size>.jpg` relative to entity's directory on filestore. To provide an alternative location, use the `entity:<icon_type>:file` *hook*.

Añadir, leer y eliminar anotaciones

Las anotaciones se pueden usar por ejemplo para llevar un seguimiento de puntuación. Para añadir una anotación a una entidad puede usar el método `annotate()` del objeto. Por ejemplo, para darle a un artículo de blog una puntuación de 5, puede utilizar:

```
$blog_post->annotate('rating', 5);
```

Para obtener las puntuaciones del artículo, use `$blogpost->getAnnotations('rating')` y si quiere eliminar una de las anotaciones puede hacerlo sobre la clase `ElggAnnotation` usando el método `$annotation->delete()`.

Para obtener una única anotación se puede usar `get_annotation()` siempre que disponga del identificador único de la anotación. Si elimina una instancia de `ElggEntity` de cualquier tipo, se eliminarán también y de manera automática todos sus metadatos, anotaciones y relaciones.

Extender ElggEntity

Si escribe una subclase de una de las clases fundamentales de Elgg, tendrá que informar a Elgg sobre cómo crear una instancia del nuevo tipo de objeto correctamente, de forma que `get_entity()` y otros métodos similares puedan devolver una clase de PHP en condiciones. Por ejemplo, si personaliza la clase `ElggGroup` en una subclase llamada `Committee` (comité), tiene que informar a Elgg sobre ella. Véase el siguiente ejemplo de extensión de una clase:

```
// Class source
class Committee extends ElggGroup {

    protected function initializeAttributes() {
        parent::initializeAttributes();
        $this->attributes['subtype'] = 'committee';
    }

    // more customizations here
}

function committee_init() {
```

(continué en la próxima página)

(proviene de la página anterior)

```
    register_entity_type('group', 'committee');

    // Tell Elgg that group subtype "committee" should be loaded using the Committee␣
↪class
    // If you ever change the name of the class, use update_subtype() to change it
    add_subtype('group', 'committee', 'Committee');
}

register_elgg_event_handler('init', 'system', 'committee_init');
```

Ahora, si llama a `get_entity()` con el identificador único de un objeto de comité, obtendrá un objeto de tipo `Committee`.

Este modelo se extrajo de la definición de `ElggFile`.

Funcionalidades avanzadas

Direcciones URL de entidades

Las direcciones URL de entidades las devuelve la interfaz `getURL()` y ofrecen a la infraestructura Elgg una forma común de dirigir a los usuarios a los manejadores de vistas apropiados para cualquier objeto.

Por ejemplo, una página de perfil en el caso de usuarios.

La dirección URL se define durante la función `elgg\_register\_entity\_url\_handler()`. La función que registre debe devolver la dirección URL apropiada para el tipo indicado, que puede ser una dirección configurada por un gestor de páginas.

El gestor predeterminado consiste en usar la interfaz de exportación predeterminada.

Entity loading performance

`elgg_get_entities` has a couple options that can sometimes be useful to improve performance.

- **preload_owners**: If the entities fetched will be displayed in a list with the owner information, you can set this option to `true` to efficiently load the owner users of the fetched entities.
- **preload_containers**: If the entities fetched will be displayed in a list using info from their containers, you can set this option to `true` to efficiently load them.
- **distinct**: When Elgg fetches entities using an SQL query, Elgg must be sure that each entity row appears only once in the result set. By default it includes a `DISTINCT` modifier on the GUID column to enforce this, but some queries naturally return unique entities. Setting the `distinct` option to false will remove this modifier, and rely on the query to enforce its own uniqueness.

The internals of Elgg entity queries is a complex subject and it's recommended to seek help on the Elgg Community site before using the `distinct` option.

Notas sobre versiones anteriores a la 1.8

update_subtype(): Esta función se introdujo en la versión 1.8. En versiones anteriores era necesario editar la base de datos manualmente en caso de haber cambiado el nombre de la clase asociada con un subtipo concreto.

elgg_register_entity_url_handler(): Esta función se introdujo en la versión 1.8. Esta nueva función substituye a register_entity_url_handler(), ahora obsoleta. Use la función vieja sólo para versiones de Elgg anteriores a la 1.8.

elgg_get_entities_from_metadata(): This function is new in 1.8. It deprecates get_entities_from_metadata(), which you should use if developing for a pre-1.8 version of Elgg.

3.9.2 Custom database functionality

It is strongly recommended to use entities wherever possible. However, Elgg supports custom SQL queries using the database API.

Example: Run SQL script on plugin activation

This example shows how you can populate your database on plugin activation.

my_plugin/activate.php:

```
if (!elgg_get_plugin_setting('database_version', 'my_plugin') {
    run_sql_script(__DIR__ . '/sql/activate.sql');
    elgg_set_plugin_setting('database_version', 1, 'my_plugin');
}
```

my_plugin/sql/activate.sql:

```
-- Create some table
CREATE TABLE prefix_custom_table(
    id INTEGER AUTO_INCREMENT,
    name VARCHAR(32),
    description VARCHAR(32),
    PRIMARY KEY (id)
);

-- Insert initial values for table
INSERT INTO prefix_custom_table (name, description)
VALUES ('Peter', 'Some guy'), ('Lisa', 'Some girl');
```

Note that Elgg execute statements through PHPs built-in functions and have limited support for comments. I.e. only single line comments are supported and must be prefixed by «– » or «# «. A comment must start at the very beginning of a line.

3.9.3 Systemlog

Nota: This section need some attention and will contain outdated information

The default Elgg system log is a simple way of recording what happens within an Elgg system. It's viewable and searchable directly from the administration panel.

System log storage

A system log row is stored whenever an event concerning an object whose class implements the *Loggable* interface is triggered. `ElggEntity` and `ElggExtender` implement *Loggable*, so a system log row is created whenever an event is performed on all objects, users, groups, sites, metadata and annotations.

Common events include:

- create
- update
- delete
- login

Creating your own system log

There are some reasons why you might want to create your own system log. For example, you might need to store a full copy of entities when they are updated or deleted, for auditing purposes. You might also need to notify an administrator when certain types of events occur.

To do this, you can create a function that listens to all events for all types of object:

```
register_elgg_event_handler('all','all','your_function_name');
```

Your function can then be defined as:

```
function your_function_name($object, $event) {
   if ($object instanceof Loggable) {
      ...
   }
}
```

You can then use the extra methods defined by *Loggable* to extract the information you need.

3.10 File System

3.10.1 Filestore

Location

Elgg's filestore is located in the site's `dataroot` that is configured during installation, and can be modified via site settings in Admin interface.

Directory Structure

The structure of the filestore is tied to file ownership by Elgg entities. Whenever the first file owned by an entity is written to the filestore, a directory corresponding to the entity GUID will be created within a parent bucket directory (buckets are bound to 5000 guids). E.g. files owned by user with guid 7777 will be located in `5000/7777/`.

When files are created, filenames can contain subdirectory names (often referred to as *$prefix* throughout the code). For instance, avatars of the above user, can be found under `5000/7777/profile/`.

3.10.2 File Objects

Writing Files

To write a file to the filestore, you would use an instance of `ElggFile`. Even though `ElggFile` extends *ElggObject* and can be stored as an actual Elgg entity, that is not always necessary (e.g. when writing thumbs of an image).

```
$file = new ElggFile();
$file->owner_guid = 7777;
$file->setFilename('portfolio/files/sample.txt');
$file->open('write');
$file->write('Contents of the file');
$file->close();

// to uprade this file to an entity
$file->subtype = 'file';
$file->save();
```

Reading Files

You can read file contents using instanceof of `ElggFile`.

```
// from an Elgg entity
$file = get_entity($file_guid);
readfile($file->getFilenameOnFilestore());
```

```
// arbitrary file on the filestore
$file = new ElggFile();
$file->owner_guid = 7777;
$file->setFilename('portfolio/files/sample.txt');

// option 1
$file->open('read');
$contents = $file->grabFile();
$file->close();

// option 2
$contents = file_get_contents($file->getFilenameOnFilestore());
```

Serving Files

You can serve files from filestore using `elgg_get_inline_url()` and `elgg_get_download_url()`. Both functions accept 3 arguments:

- **"file"** An instance of `ElggFile` to be served
- **"use_cookie"** If set to true, validity of the URL will be limited to current session
- **"expires"** Expiration time of the URL

You can use `use_cookie` and `expires` arguments as means of access control. For example, users avatars in most cases have a long expiration time and do not need to be restricted by current session - this will allows browsers to cache the images and file service will send appropriate `Not Modified` headers on consecutive requests.

For entities that are under Elgg's access control, you may want to use cookies to ensure that access settings are respected and users do not share download URLs with somebody else.

You can also invalidated all previously generated URLs by updating file's modified time, e.g. by using `touch()`.

Embedding Files

Please note that due to their nature inline and download URLs are not suitable for embedding. Embed URLs must be permanent, whereas inline and download URLs are volatile (bound to user session and file modification time).

To embed an entity icon, use `elgg_get_embed_url()`.

Handling File Uploads

In order to implement an action that saves a single file uploaded by a user, you can use the following approach:

```
// in your form
echo elgg_view('input/file', [
        'name' => 'upload',
        'label' => 'Select an image to upload',
        'help' => 'Only jpeg, gif and png images are supported',
]);
```

```
// in your action
$uploaded_files = elgg_get_uploaded_files('upload');
if (!$uploaded_files) {
        register_error("No file was uploaded");
        forward(REFERER);
}

$uploaded_file = array_shift($uploaded_files);
if (!$uploaded_file->isValid()) {
        $error = elgg_get_friendly_upload_error($uploaded_file->getError());
        register_error($error);
        forward(REFERER);
}

$supported_mimes = [
        'image/jpeg',
        'image/png',
        'image/gif',
];

$mime_type = ElggFile::detectMimeType($uploaded_file->getPathname(), $uploaded_file->
↪getClientMimeType());
if (!in_array($mime_type, $supported_mimes)) {
        register_error("$mime_type is not supported");
        forward(REFERER);
}

$file = new ElggFile();
$file->owner_guid = elgg_get_logged_in_user_guid();
if ($file->acceptUploadedFile($uploaded_file)) {
        $file->save();
}
```

If your file input supports multiple files, you can iterate through them in your action:

```
// in your form
echo elgg_view('input/file', [
        'name' => 'upload[]',
        'multiple' => true,
        'label' => 'Select images to upload',
]);
```

```
// in your action
foreach (elgg_get_uploaded_files('upload') as $upload) {
        $file = new ElggFile();
        $file->owner_guid = elgg_get_logged_in_user_guid();
        if ($file->acceptUploadedFile($upload)) {
                $file->save();
        }
}
```

3.11 Formularios y acciones

Los formularios y las acciones permiten crear, actualizar o eliminar contenido.

Los formularios de Elgg envían información a las acciones. Las acciones definen el comportamiento ante los datos recibidos.

Esta guía asume que usted está ya familiarizado con:

- *Complementos*
- *Vistas*
- *Internacionalización*

Contents

- *Signed URLs*

3.11.1 Registrar acciones

Las acciones deben registrarse antes de poder usarlas. Para registrar acciones, use `elgg_register_action`:

```
elgg_register_action("example", __DIR__ . "/actions/example.php");
```

El script `mod/example/actions/example.php` se ejecutará a partir de ahora cada vez que se envíe un formulario a `http://localhost/elgg/action/example`.

Advertencia: A stumbling point for many new developers is the URL for actions. The URL always uses `/action/` (singular) and never `/actions/` (plural). However, action script files are usually saved under the directory `/actions/` (plural) and always have an extension.

Permisos

De manera predeterminada, las acciones sólo están disponibles para usuarios registrados.

To make an action available to logged out users, pass `"public"` as the third parameter:

```
elgg_register_action("example", $filepath, "public");
```

To restrict an action to only administrators, pass `"admin"` for the last parameter:

```
elgg_register_action("example", $filepath, "admin");
```

Writing action files

Use la función `get_input` para obtener acceso a los parámetros de la solicitud:

```
$field = get_input('input_field_name', 'default_value');
```

Puede usar la API *Base de datos* para cargar entidades y realizar acciones sobre ellas.

To indicate a successful action, use `elgg_ok_response()`. This function accepts data that you want to make available to the client for XHR calls (this data will be ignored for non-XHR calls)

```
$user = get_entity($guid);
// do something

$action_data = [
   'entity' => $user,
   'stats' => [
       'friends' => $user->getFriends(['count' => true]);
   ],
];

return elgg_ok_response($action_data, 'Action was successful', 'url/to/forward/to');
```

To indicate an error, use `elgg_error_response()`

```
$user = elgg_get_logged_in_user_entity();
if (!$user) {
   // show an error and forward the user to the referring page
   // send 404 error code on AJAX calls
   return elgg_error_response('User not found', REFERRER, ELGG_HTTP_NOT_FOUND);
}

if (!$user->canEdit()) {
   // show an error and forward to user's profile
   // send 403 error code on AJAX calls
   return elgg_error_response('You are not allowed to perform this action', $user->
↪getURL(), ELGG_HTTP_FORBIDDEN);
}
```

Personalizar acciones

Antes de ejecutar cualquier acción, Elgg desencadena un gancho:

```
$result = elgg_trigger_plugin_hook('action', $action, null, true);
```

Donde `$action` es la acción a la que se llama. Si el gancho devuelve `false`, la acción no se llega a ejecutar.

Ejemplo: CAPTCHA

El módulo CAPTCHA usa lo siguiente para interceptar las acciones *register*‘ y `user/requestnewpassword` y las redirige a una función que comprueba el código del CAPTCHA. Si el código es correcto, la comprobación devuelve `true`, mientras que si no lo es devuelve `false`, lo que evita que se ejecute la acción asociada.

Esto se hace como se detalla a continuación:

```
elgg_register_plugin_hook_handler("action", "register", "captcha_verify_action_hook");
elgg_register_plugin_hook_handler("action", "user/requestnewpassword", "captcha_
↪verify_action_hook");

...

function captcha_verify_action_hook($hook, $entity_type, $returnvalue, $params) {
  $token = get_input('captcha_token');
  $input = get_input('captcha_input');

  if (($token) && (captcha_verify_captcha($input, $token))) {
    return true;
  }

  register_error(elgg_echo('captcha:captchafail'));

  return false;
}
```

Esto permite a un complemento extender una acción existente sin necesidad de substituir la acción por completo. En el caso de complemento CAPTCHA, esto le permite al complemento ofrecer la funcionalidad de CAPTCHA sin necesidad de reescribir toda la acción y actualizar su definición cada vez que ésta cambie en Elgg.

3.11.2 Actions available in core

entity/delete

If your plugin does not implement any custom logic when deleting an entity, you can use bundled delete action

```
$guid = 123;
// You can provide optional forward path as a URL query parameter
$forward_url = 'path/to/forward/to';
echo elgg_view('output/url', array(
   'text' => elgg_echo('delete'),
   'href' => "action/entity/delete?guid=$guid&forward_url=$forward_url",
   'confirm' => true,
));
```

You can customize the success message keys for your entity type and subtype, using `"entity:delete:$type:$subtype:success"` and `"entity:delete:$type:success"` keys.

```
// to add a custom message when a blog post or file is deleted
// add the translations keys in your language files
return array(
   'entity:delete:object:blog:success' => 'Blog post has been deleted,
   'entity:delete:object:file:success' => 'File titled %s has been deleted',
);
```

3.11.3 Forms

Para mostrar un formulario, utilice `elgg_view_form()` de la siguiente manera:

```
echo elgg_view_form('example');
```

Doing this generates something like the following markup:

```
<form action="http://localhost/elgg/action/example">
  <fieldset>
    <input type="hidden" name="__elgg_ts" value="1234567890" />
    <input type="hidden" name="__elgg_token" value="3874acfc283d90e34" />
  </fieldset>
</form>
```

Elgg hace algunas cosas de manera automática por usted cuando genera formularios de esta manera:

1. It sets the action to the appropriate URL based on the name of the action you pass to it
2. Añade algunos códigos aleatorios (`__elgg_ts` y `__elgg_token`) para evitar falsificaciones de peticiones entre sitios distintos, ayudando así a mantener la seguridad de las acciones.
3. Busca de manera automática el cuerpo del formulario en la vista `forms/example`.

Sitúe el contenido del formulario en la vista `forms/example` del complemento:

```
// /mod/example/views/default/forms/example.php
echo elgg_view('input/text', array('name' => 'example'));

// defer form footer rendering
// this will allow other plugins to extend forms/example view
elgg_set_form_footer(elgg_view('input/submit'));
```

Ahora, cuando ejecute `elgg_view_form('example')`, Elgg producirá lo siguiente:

```
<form action="http://localhost/elgg/action/example">
  <fieldset>
    <input type="hidden" name="__elgg_ts" value="...">
    <input type="hidden" name="__elgg_token" value="...">

    <input type="text" class="elgg-input-text" name="example">
    <div class="elgg-foot elgg-form-footer">
        <input type="submit" class="elgg-button elgg-button-submit" value="Submit">
    </div>
  </fieldset>
</form>
```

Inputs

To render a form input, use one of the bundled input views, which cover all standard HTML input elements. See individual view files for a list of accepted parameters.

```
echo elgg_view('input/select', array(
   'required' => true,
   'name' => 'status',
   'options_values' => array(
      'draft' => elgg_echo('status:draft'),
      'published' => elgg_echo('status:published'),
   ),
   // most input views will render additional parameters passed to the view
   // as tag attributes
   'data-rel' => 'blog',
));
```

The above example will render a dropdown select input:

```
<select required="required" name="status" data-rel="blog" class="elgg-input-dropdown">
   <option value="draft">Draft</option>
   <option value="published">Published</option>
</select>
```

To ensure consistency in field markup, use `elgg_view_field()`, which accepts all the parameters of the input being rendered, as well as `#label` and `#help` parameters (both of which are optional and accept HTML or text).

```
echo elgg_view_field(array(
   '#type' => 'select',
   '#label' => elgg_echo('blog:status:label'),
   '#help' => elgg_view_icon('help') . elgg_echo('blog:status:help'),
   'required' => true,
   'name' => 'status',
   'options_values' => array(
      'draft' => elgg_echo('status:draft'),
      'published' => elgg_echo('status:published'),
   ),
   'data-rel' => 'blog',
));
```

The above will generate the following markup:

```
<div class="elgg-field elgg-field-required">
   <label for="elgg-field-1" class="elgg-field-label">Blog status<span title="Required
↪" class="elgg-required-indicator">*</span></label>
   <select required="required" name="status" data-rel="blog" id="elgg-field-1" class=
↪"elgg-input-dropdown">
      <option value="draft">Draft</option>
      <option value="published">Published</option>
   </select>
   <div class="elgg-field-help elgg-text-help">
      <span class="elgg-icon-help elgg-icon"></span>This indicates whether or not the␣
↪blog is visible in the feed
   </div>
</div>
```

Input types

A list of bundled input types/views:

- `input/text` - renders a text input `<input type="text">`
- `input/plaintext` - renders a textarea `<textarea></textarea>`
- `input/longtext` - renders a WYSIWYG text input
- `input/url` - renders a url input `<input type="url">`
- `input/email` - renders an email input `<input type="email">`
- `input/checkbox` - renders a single checkbox `<input type="checkbox">`
- `input/checkboxes` - renders a set of checkboxes with the same name
- `input/radio` - renders one or more radio buttons `<input type="radio">`
- `input/submit` - renders a submit button `<input type="submit">`
- `input/button` - renders a button `<button></button>`
- `input/file` - renders a file input `<input type="file">`
- `input/select` - renders a select input `<select></select>`
- `input/hidden` - renders a hidden input `<input type="hidden">`
- `input/password` - renders a password input `<input type="password">`
- `input/number` - renders a number input `<input type="number">`
- `input/date` - renders a jQuery datepicker
- `input/access` - renders an Elgg access level select
- `input/tags` - renders an Elgg tags input
- `input/autocomplete` - renders an Elgg entity autocomplete
- `input/captcha` - placeholder view for plugins to extend
- `input/friendspicker` - renders an Elgg friend picker
- `input/userpicker` - renders an Elgg user autocomplete
- `input/location` renders an Elgg location input

3.11.4 Ficheros e imágenes

Use la vista «input/file» en la vista de contenido del formulario.

```
// /mod/example/views/default/forms/example.php
echo elgg_view('input/file', array('name' => 'icon'));
```

Elija « multipart/form-data» como el «enctype» del formulario:

```
echo elgg_view_form('example', array(
  'enctype' => 'multipart/form-data'
));
```

En el fichero de acciones, use la variable global `$_FILES` para acceder al fichero enviado:

```
$icon = $_FILES['icon']
```

3.11.5 Formularios persistentes

Los formularios persistentes son formularios que mantienen los datos introducidos por el usuario si algo evita que se puedan guardar los datos. Son «persistentes» porque los datos del usuario «persisten» en el formulario una vez enviado, a pesar de que dichos datos no han sido guardados en la base de datos. Esto mejora de manera drástica la experiencia de usuario minimizando la pérdida de datos. Elgg 1.8 incluye funciones de asistencia que le permiten convertir en persistente cualquier formulario.

Funciones de asistencia

Los formularios persistentes se añadieron a Elgg 1.8 mediante las siguientes funciones:

`elgg_make_sticky_form($name)` : Le indica al motor de Elgg que todos los campos de entrada del formulario deben ser persistentes.

`elgg_clear_sticky_form($name)` : Le indica al motor de Elgg que debe descartar todos los campos de entrada persistentes del formulario.

`elgg_is_sticky_form($name)` : Comprueba si `$name` es un formulario persistente válido.

`elgg_get_sticky_values($name)` : Devuelve todos los valores persistentes almacenados para `$name` por `elgg_make_sticky_form()`.

Resumen

El flujo básico de uso de formularios persistentes consiste en: (1) Llamar a `elgg_make_sticky_form($name)` al principio de las acciones para formularios que desee hacer persistentes, (2) usar `elgg_is_sticky_form($name)` y `elgg_get_sticky_values($name)` para obtener los valores persistidos a la hora de generar la vista del formulario y (3) llamar a `elgg_clear_sticky_form($name)` una vez la acción se completase correctamente o después de que los datos se cargasen mediante `elgg_get_sticky_values($name)`.

Ejemplo: Registro de una cuenta de usuario

Los formularios persistentes simples requieren un poco de lógica para determinar los campos de entrada del formulario. La lógica se coloca en la parte superior del cuerpo de la vista del propio formulario.

La vista del formulario de registro establece en primer lugar los valores predeterminados de los campos de entrada, y a continuación comprueba si entre ellos hay campos con valores persistidos. De haber campos con valores persistidos, el formulario carga dichos valores antes de vaciar el formulario persistente:

```
// views/default/forms/register.php
$password = $password2 = '';
$username = get_input('u');
$email = get_input('e');
$name = get_input('n');

if (elgg_is_sticky_form('register')) {
    extract(elgg_get_sticky_values('register'));
    elgg_clear_sticky_form('register');
}
```

La acción de registro crea el formulario persistente y lo vacía una vez se completa la acción:

```
// actions/register.php
elgg_make_sticky_form('register');

...

$guid = register_user($username, $password, $name, $email, false, $friend_guid,
↪$invitecode);

if ($guid) {
    elgg_clear_sticky_form('register');
    ....
}
```

Ejemplo: Marcadores

La acción y formulario de guardado incluidos en el complemento «Marcadores» son un ejemplo de un formulario persistente complejo.

La vista de formulario para la acción de guardar un marcador usa `elgg_extract()` para obtener valores del vector `$vars`:

```
// mod/bookmarks/views/default/forms/bookmarks/save.php
$title = elgg_extract('title', $vars, '');
$desc = elgg_extract('description', $vars, '');
$address = elgg_extract('address', $vars, '');
$tags = elgg_extract('tags', $vars, '');
$access_id = elgg_extract('access_id', $vars, ACCESS_DEFAULT);
$container_guid = elgg_extract('container_guid', $vars);
$guid = elgg_extract('guid', $vars, null);
$shares = elgg_extract('shares', $vars, array());
```

Los scripts del gestor de páginas prepara las variables del formulario y llama a `elgg_view_form()` pasándole los valores correctos:

```
// mod/bookmarks/pages/add.php
$vars = bookmarks_prepare_form_vars();
$content = elgg_view_form('bookmarks/save', array(), $vars);
```

De manera semejante, `mod/bookmarks/pages/edit.php` usa la misma función, pero le pasa la entidad que se está editando como argumento:

```
$bookmark_guid = get_input('guid');
$bookmark = get_entity($bookmark_guid);

...

$vars = bookmarks_prepare_form_vars($bookmark);
$content = elgg_view_form('bookmarks/save', array(), $vars);
```

El fichero de la biblioteca define `bookmarks_prepare_form_vars()`. Esta función acepta una instancia de `ElggEntity` como argumento y hace 3 cosas:

1. Define los nombres de los campos de entrada y sus valores predeterminados.
2. Extrae los valores de un objeto de marcador si lo recibe.
3. Extrae los valores de un formulario persistente si éste existe.

Por hacer: incluir directamente desde «lib/bookmarks.php».

```
// mod/bookmarks/lib/bookmarks.php
function bookmarks_prepare_form_vars($bookmark = null) {
     // input names => defaults
  $values = array(
    'title' => get_input('title', ''), // bookmarklet support
    'address' => get_input('address', ''),
    'description' => '',
    'access_id' => ACCESS_DEFAULT,
    'tags' => '',
    'shares' => array(),
    'container_guid' => elgg_get_page_owner_guid(),
    'guid' => null,
    'entity' => $bookmark,
  );

  if ($bookmark) {
       foreach (array_keys($values) as $field) {
       if (isset($bookmark->$field)) {
         $values[$field] = $bookmark->$field;
       }
    }
  }

  if (elgg_is_sticky_form('bookmarks')) {
       $sticky_values = elgg_get_sticky_values('bookmarks');
       foreach ($sticky_values as $key => $value) {
      $values[$key] = $value;
    }
  }

  elgg_clear_sticky_form('bookmarks');

  return $values;
}
```

La acción de guardar comprueba los campos de entrada, y luego vacía el formulario persistente cuando se completa correctamente:

```
// mod/bookmarks/actions/bookmarks/save.php
elgg_make_sticky_form('bookmarks');
```

(continué en la próxima página)

(proviene de la página anterior)

```
...

if ($bookmark->save()) {
     elgg_clear_sticky_form('bookmarks');
}
```

3.11.6 AJAX

See the *Ajax guide* for instructions on calling actions from JavaScript.

3.11.7 Seguridad

For enhanced security, all actions require an CSRF token. Calls to action URLs that do not include security tokens will be ignored and a warning will be generated.

Algunas vistas y funciones generan códigos aleatorios de seguridad de forma automática:

```
elgg_view('output/url', array('is_action' => TRUE));
elgg_view('input/securitytoken');
$url = elgg_add_action_tokens_to_url("http://localhost/elgg/action/example");
```

En algunos casos excepcionales, puede que necesite generar esos códigos manualmente:

```
$__elgg_ts = time();
$__elgg_token = generate_action_token($__elgg_ts);
```

También puede acceder a los códigos de seguridad desde JavaScript:

```
elgg.security.token.__elgg_ts;
elgg.security.token.__elgg_token;
```

Éstos se actualizan de manera periódica, por lo que deberían estar siempre al día.

3.11.8 Security Tokens

On occasion we need to pass data through an untrusted party or generate an «unguessable token» based on some data. The industry-standard HMAC algorithm is the right tool for this. It allows us to verify that received data were generated by our site, and were not tampered with. Note that even strong hash functions like SHA-2 should *not* be used without HMAC for these tasks.

Elgg provides `elgg_build_hmac()` to generate and validate HMAC message authentication codes that are unguessable without the site's private key.

```
// generate a querystring such that $a and $b can't be altered
$a = 1234;
$b = "hello";
$query = http_build_query([
    'a' => $a,
    'b' => $b,
    'mac' => elgg_build_hmac([$a, $b])->getToken(),
]);
$url = "action/foo?$query";
```

(continué en la próxima página)

(proviene de la página anterior)

```
// validate the querystring
$a = (int) get_input('a', '', false);
$b = (string) get_input('b', '', false);
$mac = get_input('mac', '', false);

if (elgg_build_hmac([$a, $b])->matchesToken($mac)) {
    // $a and $b have not been altered
}
```

Note: If you use a non-string as HMAC data, you must use types consistently. Consider the following:

```
$mac = elgg_build_hmac([123, 456])->getToken();

// type of first array element differs
elgg_build_hmac(["123", 456])->matchesToken($mac); // false

// types identical to original
elgg_build_hmac([123, 456])->matchesToken($mac); // true
```

3.11.9 Signed URLs

Signed URLs offer a limited level of security for situations where action tokens are not suitable, for example when sending a confirmation link via email. URL signatures verify that the URL has been generated by your Elgg installation (using site secret) and that the URL query elements were not tampered with.

URLs a signed with an unguessable SHA-256 HMAC key. See *Security Tokens* for more details.

```
$url = elgg_http_add_url_query_element(elgg_normalize_url('confirm'), [
   'user_guid' => $user_guid,
]);

$url = elgg_http_get_signed_url($url);

notify_user($user_guid, $site->guid, 'Confirm', "Please confirm by clicking this␣
↪link: $url");
```

Advertencia: Signed URLs do not offer CSRF protection and should not be used instead of action tokens.

3.12 Funciones de asistencia

3.12.1 Input and output

- `get_input($name)` Grabs information from a form field (or any variable passed using GET or POST). Also sanitises input, stripping Javascript etc.
- `set_input($name, $value)` Forces a value to a particular variable for subsequent retrieval by `get_input()`

3.12.2 Entity methods

- `$entity->getURL()` Returns the URL of any entity in the system
- `$entity->getGUID()` Returns the GUID of any entity in the system
- `$entity->canEdit()` Returns whether or not the current user can edit the entity
- `$entity->getOwnerEntity()` Returns the ElggUser owner of a particular entity

3.12.3 Entity and context retrieval

- `elgg_get_logged_in_user_entity()` Returns the ElggUser for the current user
- `elgg_get_logged_in_user_guid()` Returns the GUID of the current user
- `elgg_is_logged_in()` Is the viewer logged in
- `elgg_is_admin_logged_in()` Is the view an admin and logged in
- `elgg_gatekeeper()` Shorthand for checking if a user is logged in. Forwards user to front page if not
- `elgg_admin_gatekeeper()` Shorthand for checking the user is logged in and is an admin. Forwards user to front page if not
- `get_user($user_guid)` Given a GUID, returns a full ElggUser entity
- `elgg_get_page_owner_guid()` Returns the GUID of the current page owner, if there is one
- `elgg_get_page_owner_entity()` Like elgg_get_page_owner_guid() but returns the full entity
- `elgg_get_context()` Returns the current page's context - eg «blog» for the blog plugin, «thewire» for the wire, etc. Returns «main» as default
- `elgg_set_context($context)` Forces the context to be a particular value
- `elgg_push_context($context)` Adds a context to the stack
- `elgg_pop_context()` Removes the top context from the stack
- `elgg_in_context($context)` Checks if you're in a context (this checks the complete stack, eg. "widget" in "groups")

3.12.4 Complementos

- `elgg_is_active_plugin($plugin_id)` Check if a plugin is installed and enabled

3.12.5 Interface and annotations

- `elgg_view_image_block($icon, $info)` Return the result in a formatted list
- `elgg_view_comments($entity)` Returns any comments associated with the given entity
- `elgg_get_friendly_time($unix_timestamp)` Returns a date formatted in a friendlier way - «18 minutes ago», «2 days ago», etc.
- You can pass `'use_hover' => false` to the user icon view if you don't want the avatar drop down menu to appear e.g.

```
elgg_view_entity_icon($user, 'small', array('use_hover' => false));
```

3.13 Internacionalización

Haz posible traducir la interfaz de tu sitio a cualquier idioma.

Si quiere colaborar en la traducción de Elgg a algún idioma, échele un ojo a la guía para colaboradores.

The default language is `en` for English. Currently Elgg will always fall back to an English translation, even if the site's language is not English; this is a known bug.

3.13.1 Resumen

Las traducciones se almacenan en ficheros PHP dentro de la carpeta `/languages` del complemento. Cada fichero corresponde a un idioma. El formato es `/languages/<código del idioma>.php` donde `<código del idioma>` es el código ISO 639-1 del idioma. Por ejemplo:

```
<?php // mod/example/languages/en.php

return [
        'example:text' => 'Some example text',
];
```

To override an existing translation, include it in your plugin's language file, and make sure your plugin is ordered later on the Admin > Plugins page:

```
<?php // mod/better_example/languages/en.php

return [
        'example:text' => 'Some better text!',
];
```

Nota: Unless you are overriding core's or another plugin's language strings, it is good practice for the language keys to start with your plugin name. For example: `yourplugin:success`, `yourplugin:title`, etc. This helps avoid conflicts with other language keys.

3.13.2 API del lado del servidor

`elgg_echo($clave, $argumentos, $idioma)`

Imprimir la traducción de la clave al idioma actual.

Ejemplo:

```
echo elgg_echo('example:text');
```

También permite la substitución de variables mediante la sintaxis de `sprintf`:

```
// 'welcome' => 'Welcome to %s, %s!'
echo elgg_echo('welcome', [
        elgg_get_config('sitename'),
        elgg_get_logged_in_user_entity()->name,
]);
```

Para especificar un idioma concreto al que traducir el texto indicado, use el tercer parámetro:

```
echo elgg_echo('welcome', [], $user->language);
```

To first test whether `elgg_echo()` can find a translation:

```
$key = 'key:that:might:not:exist';
if (!elgg_language_key_exists($key)) {
        $key = 'fallback:key';
}

echo elgg_echo($key);
```

Nota: Some APIs allow creating translations for new keys. Translators should always include an English translation as a fallback. This makes `elgg_language_key_exists($key)` a reliable way to predict whether `elgg_echo($key)` will succeed.

3.13.3 API de JavaScript

`elgg.echo(key, args)`

This function is like `elgg_echo` in PHP.

Client-side translations are loaded asynchronously. Ensure translations are available by requiring the «elgg» AMD module:

```
define(function(require) {
        var elgg = require("elgg");

        alert(elgg.echo('my_key'));
});
```

Translations are also available after the `init, system` JavaScript event.

3.14 JavaScript

Contents

3.14.1 AMD

Developers should use the AMD (Asynchronous Module Definition) standard for writing JavaScript code in Elgg.

Here we'll describe making and executing AMD modules. The RequireJS documentation for defining modules may also be of use.

Executing a module in the current page

Telling Elgg to load an existing module in the current page is easy:

```
<?php
elgg_require_js("myplugin/say_hello");
```

On the client-side, this will asynchronously load the module, load any dependencies, and execute the module's definition function, if it has one.

Defining the Module

Here we define a basic module that alters the page, by passing a «definition function» to `define()`:

```
// in views/default/myplugin/say_hello.js

define(function(require) {
    var elgg = require("elgg");
    var $ = require("jquery");

    $('body').append(elgg.echo('hello_world'));
});
```

The module's name is determined by the view name, which here is `myplugin/say_hello.js`. We strip the `.js` extension, leaving `myplugin/say_hello`.

Advertencia: The definition function **must** have one argument named `require`.

Making modules dependent on other modules

Below we refactor a bit so that the module depends on a new `myplugin/hello` module to provide the greeting:

```
// in views/default/myplugin/hello.js

define(function(require) {
    var elgg = require("elgg");

    return elgg.echo('hello_world');
});
```

```
// in views/default/myplugin/say_hello.js

define(function(require) {
    var $ = require("jquery");
    var hello = require("myplugin/hello");

    $('body').append(hello);
});
```

Passing settings to modules

The `elgg.data` plugin hooks

The `elgg` module provides an object `elgg.data` which is populated from two server side hooks:

- **elgg.data, site**: This filters an associative array of site-specific data passed to the client and cached.
- **elgg.data, page**: This filters an associative array of uncached, page-specific data passed to the client.

Let's pass some data to a module:

```
<?php

function myplugin_config_site($hook, $type, $value, $params) {
    // this will be cached client-side
    $value['myplugin']['api'] = elgg_get_site_url() . 'myplugin-api';
    $value['myplugin']['key'] = 'none';
```

(continué en la próxima página)

(proviene de la página anterior)

```
    return $value;
}

function myplugin_config_page($hook, $type, $value, $params) {
    $user = elgg_get_logged_in_user_entity();
    if ($user) {
        $value['myplugin']['key'] = $user->myplugin_api_key;
        return $value;
    }
}

elgg_register_plugin_hook_handler('elgg.data', 'site', 'myplugin_config_site');
elgg_register_plugin_hook_handler('elgg.data', 'page', 'myplugin_config_page');
```

```
define(function(require) {
    var elgg = require("elgg");

    var api = elgg.data.myplugin.api;
    var key = elgg.data.myplugin.key; // "none" or a user's key

    // ...
});
```

Nota: In `elgg.data`, page data overrides site data. Also note `json_encode()` is used to copy data client-side, so the data must be JSON-encodable.

Making a config module

You can use a PHP-based module to pass values from the server. To make the module `myplugin/settings`, create the view file `views/default/myplugin/settings.js.php` (note the double extension `.js.php`).

```
<?php

// this will be cached client-side
$settings = [
    'api' => elgg_get_site_url() . 'myplugin-api',
    'key' => null,
];
?>
define(<?php echo json_encode($settings); ?>);
```

You must also manually register the view as an external resource:

```
<?php
// note the view name does not include ".php"
elgg_register_simplecache_view('myplugin/settings.js');
```

Nota: The PHP view is cached, so you should treat the output as static (the same for all users) and avoid session-specific logic.

Setting the URL of a module

You may have an AMD script outside your views you wish to make available as a module.

The best way to accomplish this is by configuring the path to the file using the `views.php` file in the root of your plugin:

```
<?php // views.php
return [
    'default' => [
        'underscore.js' => 'vendor/bower-asset/underscore/underscore.min.js',
    ],
];
```

If you've copied the script directly into your plugin instead of managing it with Composer, you can use something like this instead:

```
<?php // views.php
return [
    'default' => [
        'underscore.js' => __DIR__ . '/bower_components/underscore/underscore.min.js',
    ],
];
```

That's it! Elgg will now load this file whenever the «underscore» module is requested.

Using traditional JS libraries as modules

It's possible to support JavaScript libraries that do not declare themselves as AMD modules (i.e. they declare global variables instead) if you shim them by setting `exports` and `deps` in `elgg_define_js`:

```
// set the path, define its dependencies, and what value it returns
elgg_define_js('jquery.form', [
    'deps' => ['jquery'],
    'exports' => 'jQuery.fn.ajaxForm',
]);
```

When this is requested client-side:

1. The jQuery module is loaded, as it's marked as a dependency.
2. `https://elgg.example.org/cache/125235034/views/default/jquery.form.js` is loaded and executed.
3. The value of `window.jQuery.fn.ajaxForm` is returned by the module.

Advertencia: Las llamadas a `elgg_define_js()` deben estar en un manejador del evento `init, system`.

Cosas a tener en cuenta

1. Do not use `elgg.provide()` anymore nor other means to attach code to `elgg` or other global objects. Use modules.
2. Return the value of the module instead of adding to a global variable.
3. Static (.js,.css,etc.) files are automatically minified and cached by Elgg's simplecache system.

4. The configuration is also cached in simplecache, and should not rely on user-specific values like `get_language()`.

3.14.2 Booting your plugin

To add functionality to each page, or make sure your hook handlers are registered early enough, you may create a boot module for your plugin, with the name `boot/<plugin_id>`.

```
// in views/default/boot/example.js

define(function(require) {
    var elgg = require("elgg");
    var Plugin = require("elgg/Plugin");

    // plugin logic
    function my_init() { ... }

    return new Plugin({
        // executed in order of plugin priority
        init: function () {
            elgg.register_hook_handler("init", "system", my_init, 400);
        }
    });
});
```

When your plugin is active, this module will automatically be loaded on each page. Other modules can depend on `elgg/init` to make sure all boot modules are loaded.

Each boot module **must** return an instance of `elgg/Plugin`. The constructor must receive an object with a function in the `init` key. The `init` function will be called in the order of the plugin in Elgg's admin area.

Nota: Though not strictly necessary, you may want to use the `init, system` event to control when your initialization code runs with respect to other modules.

Advertencia: A boot module **cannot** depend on the modules `elgg/init` or `elgg/ready`.

3.14.3 Modules provided with Elgg

Modules `jquery` and `jquery-ui`

You must depend on these modules to use `$` or `$.ui` methods. In the future Elgg may stop loading these by default.

Module `elgg`

`elgg.echo()`

Traducir texto de la interfaz.

```
elgg.echo('example:text', ['arg1']);
```

`elgg.system_message()`

Mostrar un mensaje de estado al usuario.

```
elgg.system_message(elgg.echo('success'));
```

`elgg.register_error()`

Mostrar un mensaje de error al usuario.

```
elgg.register_error(elgg.echo('error'));
```

`elgg.normalize_url()`

Modificar una URL para que sea relativa a la raíz del sitio Elgg:

```
// "http://localhost/elgg/blog"
elgg.normalize_url('/blog');
```

`elgg.forward()`

Redirigir a una página nueva.

```
elgg.forward('/blog');
```

Esta función normaliza de manera automática la URL.

`elgg.parse_url()`

Analiza una URL y divídela en partes:

```
// returns {
//   fragment: "fragment",
//   host: "community.elgg.org",
//   path: "/file.php",
//   query: "arg=val"
// }
elgg.parse_url('http://community.elgg.org/file.php?arg=val#fragment');
```

`elgg.get_page_owner_guid()`

Obtener el identificador único del propietario de la página actual.

`elgg.register_hook_handler()`

Register a hook handler with the event system. For best results, do this in a plugin boot module.

```
// boot module: /views/default/boot/example.js
define(function (require) {
    var elgg = require('elgg');
    var Plugin = require('elgg/Plugin');

    elgg.register_hook_handler('foo', 'bar', function () { ... });

    return new Plugin();
});
```

`elgg.trigger_hook()`

Emit a hook event in the event system. For best results depend on the elgg/init module.

```
// old
value = elgg.trigger_hook('my_plugin:filter', 'value', {}, value);

define(function (require) {
    require('elgg/init');
    var elgg = require('elgg');

    value = elgg.trigger_hook('my_plugin:filter', 'value', {}, value);
});
```

`elgg.security.refreshToken()`

Obligar a que se actualicen todos los códigos aleatorios de la página que se usan para dificultar los ataques de falsificación de peticiones entre sitios.

De manera predeterminada, esto se ejecuta de manera automática cada 5 minutos.

Para esto hace falta un código de seguridad correcto en la versión 1.8 y anteriores, pero no en la versión 1.9 y posteriores.

El usuario recibirá un aviso si la sesión ha caducado.

`elgg.security.addToken()`

Añadir un código aleatorio de seguridad a un objeto, URL, o texto de consulta:

```
// returns {
//   __elgg_token: "1468dc44c5b437f34423e2d55acfdd87",
//   __elgg_ts: 1328143779,
//   other: "data"
// }
elgg.security.addToken({'other': 'data'});

// returns: "action/add?__elgg_ts=1328144079&__elgg_
↪token=55fd9c2d7f5075d11e722358afd5fde2"
elgg.security.addToken("action/add");

// returns "?arg=val&__elgg_ts=1328144079&__elgg_
↪token=55fd9c2d7f5075d11e722358afd5fde2"
elgg.security.addToken("?arg=val");
```

`elgg.get_logged_in_user_entity()`

Devuelve el usuario actual como un objeto ElggUser de JavaScript.

`elgg.get_logged_in_user_guid()`

Devuelve el identificador único del usuario actual.

`elgg.is_logged_in()`

Devuelve `true` si el usuario está identificado.

`elgg.is_admin_logged_in()`

Devuelve `true` si el usuario está identificado como administrador.

`elgg.config.get_language()`

Devuelve el idioma de la página actual.

En el objeto `elgg` existen ciertos valores de configuración:

```
// The root of the website.
elgg.config.wwwroot;
// The default site language.
elgg.config.language;
// The current page's viewtype
elgg.config.viewtype;
// The Elgg version (YYYYMMDDXX).
elgg.config.version;
// The Elgg release (X.Y.Z).
elgg.config.release;
```

Module `elgg/Ajax`

See the *AJAX* page for details.

Module `elgg/init`

`elgg/init` loads and initializes all boot modules in priority order and triggers the [init, system] hook.

Require this module to make sure all plugins are ready.

Module `elgg/Plugin`

Used to create a *boot module*.

Module `elgg/ready`

`elgg/ready` loads and initializes all plugin boot modules in priority order.

Require this module to make sure all plugins are ready.

Module `elgg/spinner`

The `elgg/spinner` module can be used to create an Ajax loading indicator fixed to the top of the window.

```
define(function (require) {
   var spinner = require('elgg/spinner');

   elgg.action('friend/add', {
       beforeSend: spinner.start,
       complete: spinner.stop,
       success: function (json) {
           // ...
       }
   });
});
```

Nota: The `elgg/Ajax` module uses the spinner by default.

Module `elgg/popup`

The `elgg/popup` module can be used to display an overlay positioned relatively to its anchor (trigger).

The `elgg/popup` module is loaded by default, and binding a popup module to an anchor is as simple as adding `rel="popup"` attribute and defining target module with a `href` (or `data-href`) attribute. Popup module positioning can be defined with `data-position` attribute of the trigger element.

```
echo elgg_format_element('div', [
   'class' => 'elgg-module-popup hidden',
   'id' => 'popup-module',
], 'Popup module content');

// Simple anchor
echo elgg_view('output/url', [
   'href' => '#popup-module',
   'text' => 'Show popup',
   'rel' => 'popup',
]);

// Button with custom positioning of the popup
echo elgg_format_element('button', [
   'rel' => 'popup',
   'class' => 'elgg-button elgg-button-submit',
   'text' => 'Show popup',
   'data-href' => '#popup-module',
   'data-position' => json_encode([
      'my' => 'center bottom',
      'at' => 'center top',
   ]),
]);
```

The `elgg/popup` module allows you to build out more complex UI/UX elements. You can open and close popup modules programmatically:

```
define(function(require) {
   var $ = require('jquery');
   $(document).on('click', '.elgg-button-popup', function(e) {

      e.preventDefault();

      var $trigger = $(this);
      var $target = $('#my-target');
              var $close = $target.find('.close');

      require(['elgg/popup'], function(popup) {
                popup.open($trigger, $target, {
                       'collision': 'fit none'
                });

        $close.on('click', popup.close);
              });
   });
});
```

You can use `getOptions, ui.popup` plugin hook to manipulate the position of the popup before it has been opened. You can use jQuery `open` and `close` events to manipulate popup module after it has been opened or closed.

```
define(function(require) {

   var elgg = require('elgg');
   var $ = require('jquery');

   $('#my-target').on('open', function() {
      var $module = $(this);
      var $trigger = $module.data('trigger');

      elgg.ajax('ajax/view/my_module', {
         beforeSend: function() {
            $trigger.hide();
            $module.html('').addClass('elgg-ajax-loader');
         },
         success: function(output) {
            $module.removeClass('elgg-ajax-loader').html(output);
         }
      });
   }).on('close', function() {
      var $trigger = $(this).data('trigger');
      $trigger.show();
   });
});
```

Open popup modules will always contain the following data that can be accessed via `$.data()`:

- `trigger` - jQuery element used to trigger the popup module to open
- `position` - An object defining popup module position that was passed to `$.position()`

By default, `target` element will be appended to `$('body')` thus altering DOM hierarchy. If you need to preserve the DOM position of the popup module, you can add `.elgg-popup-inline` class to your trigger.

Module `elgg/widgets`

Plugins that load a widget layout via Ajax should initialize via this module:

```
require(['elgg/widgets'], function (widgets) {
    widgets.init();
});
```

Module `elgg/lightbox`

Elgg is distributed with the Colorbox jQuery library. Please go to http://www.jacklmoore.com/colorbox for more information on the options of this lightbox.

Use the following classes to bind your anchor elements to a lightbox:

- `elgg-lightbox` - loads an HTML resource
- `elgg-lightbox-photo` - loads an image resource (should be used to avoid displaying raw image bytes instead of an `img` tag)
- `elgg-lightbox-inline` - displays an inline HTML element in a lightbox
- `elgg-lightbox-iframe` - loads a resource in an `iframe`

You may apply colorbox options to an individual `elgg-lightbox` element by setting the attribute `data-colorbox-opts` to a JSON settings object.

```
echo elgg_view('output/url', [
   'text' => 'Open lightbox',
   'href' => 'ajax/view/my_view',
   'class' => 'elgg-lightbox',
   'data-colorbox-opts' => json_encode([
      'width' => '300px',
   ])
]);
```

Use `"getOptions", "ui.lightbox"` plugin hook to filter options passed to `$.colorbox()` whenever a lightbox is opened. Note that the hook handler should depend on `elgg/init` AMD module.

`elgg/lightbox` AMD module should be used to open and close the lightbox programmatically:

```
define(function(require) {
   var lightbox = require('elgg/lightbox');
   var spinner = require('elgg/spinner');

   lightbox.open({
      html: '<p>Hello world!</p>',
      onClosed: function() {
         lightbox.open({
            onLoad: spinner.start,
            onComplete: spinner.stop,
            photo: true,
            href: 'https://elgg.org/cache/1457904417/default/community_theme/graphics/
↪logo.png',
         });
      }
   });
});
```

To support gallery sets (via `rel` attribute), you need to bind colorbox directly to a specific selector (note that this will ignore `data-colorbox-opts` on all elements in a set):

```
require(['elgg/lightbox'], function(lightbox) {
   var options = {
      photo: true,
      width: 500
   };
   lightbox.bind('a[rel="my-gallery"]', options, false); // 3rd attribute ensures␣
↪binding is done without proxies
});
```

You can also resize the lightbox programmatically if needed:

```
define(function(require) {
   var lightbox = require('elgg/lightbox');

   lightbox.resize({
      width: '300px'
   });
});
```

Module `elgg/ckeditor`

This module can be used to add WYSIWYG editor to a textarea (requires `ckeditor` plugin to be enabled). Note that WYSIWYG will be automatically attached to all instances of `.elgg-input-longtext`.

```
require(['elgg/ckeditor'], function (elggCKEditor) {
   elggCKEditor.bind('#my-text-area');

   // Toggle CKEditor
   elggCKEditor.toggle('#my-text-area');

   // Focus on CKEditor input
   elggCKEditor.focus('#my-text-area');
   // or
   $('#my-text-area').trigger('focus');

   // Reset CKEditor input
   elggCKEditor.reset('#my-text-area');
   // or
   $('#my-text-area').trigger('reset');

});
```

Inline tabs component

Inline tabs component fires an `open` event whenever a tabs is open and, in case of ajax tabs, finished loading:

```
// Add custom animation to tab content
    require(['jquery', 'elgg/ready'], function($) {
            $(document).on('open', '.theme-sandbox-tab-callback', function() {
                    $(this).find('a').text('Clicked!');
                    $(this).data('target').hide().show('slide', {
                            duration: 2000,
                            direction: 'right',
                            complete: function() {
                                    alert('Thank you for clicking. We hope you␣
↪enjoyed the show!');
                                    $(this).css('display', ''); // .show() adds␣
↪display property
                            }
                    });
            });
    });
```

3.14.4 Traditional scripts

Although we highly recommend using AMD modules, you can register scripts with `elgg_register_js`:

```
elgg_register_js('jquery', $cdnjs_url);
```

Esto substituirá cualquier URL previamente registrada bajo este nombre.

Para cargar una biblioteca en la página actual, use `elgg_load_js`:

```
elgg_load_js('jquery');
```

This will load the library in the page footer. You must use the `require()` function to depend on modules like `elgg` and `jquery`.

Advertencia:

Using inline scripts is NOT SUPPORTED because:

- No es posible escribir pruebas para ellos, lo cual dificulta mantenerlos.
- No se pueden guardar en caché, lo cual afecta al rendimiento.
- They prevent use of Content-Security-Policy (security)
- They prevent scripts from being loaded with `defer` or `async` (performance)

Los scripts in situ en el núcleo de Elgg o en los complementos que se incluyen junto con el núcleo están considerados código viejo pendiente de renovar.

3.14.5 Hooks

The JS engine has a hooks system similar to the PHP engine's plugin hooks: hooks are triggered and plugins can register functions to react or alter information. There is no concept of Elgg events in the JS engine; everything in the JS engine is implemented as a hook.

Registering hook handlers

Handler functions are registered using `elgg.register_hook_handler()`. Multiple handlers can be registered for the same hook.

The following example registers the `handleFoo` function for the `foo, bar` hook.

```
define(function (require) {
    var elgg = require('elgg');
    var Plugin = require('elgg/Plugin');

    function handleFoo(hook, type, params, value) {
        // do something
    }

    elgg.register_hook_handler('foo', 'bar', handleFoo);

    return new Plugin();
});
```

The handler function

The handler will receive 4 arguments:

- **hook** - The hook name
- **type** - The hook type
- **params** - An object or set of parameters specific to the hook

- **value** - The current value

The `value` will be passed through each hook. Depending on the hook, callbacks can simply react or alter data.

Triggering custom hooks

Plugins can trigger their own hooks:

```
define(function(require) {
    require('elgg/init');
    var elgg = require('elgg');

    elgg.trigger_hook('name', 'type', {params}, "value");
});
```

Nota: Be aware of timing. If you don't depend on elgg/init, other plugins may not have had a chance to register their handlers.

Available hooks

init, system Plugins should register their init functions for this hook. It is fired after Elgg's JS is loaded and all plugin boot modules have been initialized. Depend on the `elgg/init` module to be sure this has completed.

ready, system This hook is fired when the system has fully booted (after init). Depend on the `elgg/ready` module to be sure this has completed.

getOptions, ui.popup This hook is fired for pop up displays (`"rel"="popup"`) and allows for customized placement options.

getOptions, ui.lightbox This hook can be used to filter options passed to `$.colorbox()`

config, ckeditor This filters the CKEditor config object. Register for this hook in a plugin boot module. The defaults can be seen in the module `elgg/ckeditor/config`.

prepare, ckeditor This hook can be used to decorate `CKEDITOR` global. You can use this hook to register new CKEditor plugins and add event bindings.

ajax_request_data, * This filters request data sent by the `elgg/Ajax` module. See *AJAX* for details.

ajax_response_data, * This filters the response data returned to users of the `elgg/Ajax` module. See *AJAX* for details.

insert, editor This hook is triggered by the embed plugin and can be used to filter content before it is inserted into the textarea. This hook can also be used by WYSIWYG editors to insert content using their own API (in this case the handler should return `false`). See ckeditor plugin for an example.

3.14.6 Third-party assets

We recommend managing third-party scripts and styles with Composer. Elgg core uses `fxp/composer-asset-plugin` for this purpose. This plugin allows you to pull dependencies from the Bower or NPM package repositories, but using the Composer command-line tool.

For example, to include jQuery, you could run the following Composer commands:

```
composer global require fxp/composer-asset-plugin:~1.1.4
composer require bower-asset/jquery:~2.0
```

Nota: `fxp/composer-asset-plugin` must be installed globally! See https://github.com/francoispluchino/composer-asset-plugin for more info.

3.15 Menus

Elgg contains helper code to build menus throughout the site.

Every single menu requires a name, as does every single menu item. These are required in order to allow easy overriding and manipulation, as well as to provide hooks for theming.

Contents

3.15.1 Basic usage

Basic functionalities can be achieved through these two functions:

- elgg_register_menu_item() to add an item to a menu
- elgg_unregister_menu_item() to remove an item from a menu

You normally want to call them from your plugin's init function.

Examples

```
// Add a new menu item to the site main menu
elgg_register_menu_item('site', array(
        'name' => 'itemname',
        'text' => 'This is text of the item',
        'href' => '/item/url',
));
```

```
// Remove the "Elgg" logo from the topbar menu
elgg_unregister_menu_item('topbar', 'elgg_logo');
```

3.15.2 Advanced usage

You can get more control over menus by using *plugin hooks* and the public methods provided by the ElggMenuItem class.

There are two hooks that can be used to modify a menu:

- `'register', 'menu:<menu name>'` to add or modify items (especially in dynamic menus)
- `'prepare', 'menu:<menu name>'` to modify the structure of the menu before it is displayed

When you register a plugin hook handler, replace the `<menu name>` part with the internal name of the menu.

The third parameter passed into a menu handler contains all the menu items that have been registered so far by Elgg core and other enabled plugins. In the handler we can loop through the menu items and use the class methods to interact with the properties of the menu item.

Examples

Example 1: Change the URL for menu item called «albums» in the `owner_block` menu:

```
/**
 * Initialize the plugin
 */
function my_plugin_init() {
        // Register a plugin hook handler for the owner_block menu
        elgg_register_plugin_hook_handler('register', 'menu:owner_block', 'my_owner_
↪block_menu_handler');
}

/**
 * Change the URL of the "Albums" menu item in the owner_block menu
 */
function my_owner_block_menu_handler($hook, $type, $items, $params) {
        $owner = $params['entity'];

        // Owner can be either user or a group, so we
        // need to take both URLs into consideration:
        switch ($owner->getType()) {
                case 'user':
                        $url = "album/owner/{$owner->guid}";
                        break;
                case 'group':
                        $url = "album/group/{$owner->guid}";
                        break;
        }

        foreach ($items as $key => $item) {
                if ($item->getName() == 'albums') {
                        // Set the new URL
                        $item->setURL($url);
                        break;
                }
        }

        return $items;
}
```

Example 2: Modify the `entity` menu for the `ElggBlog` objects

- Remove the thumb icon
- Change the «Edit» text into a custom icon

```
/**
 * Initialize the plugin
 */
function my_plugin_init() {
        // Register a plugin hook handler for the entity menu
        elgg_register_plugin_hook_handler('register', 'menu:entity', 'my_entity_menu_
↪handler');
}

/**
 * Customize the entity menu for ElggBlog objects
 */
function my_entity_menu_handler($hook, $type, $items, $params) {
        // The entity can be found from the $params parameter
        $entity = $params['entity'];

        // We want to modify only the ElggBlog objects, so we
        // return immediately if the entity is something else
        if (!$entity instanceof ElggBlog) {
                return $menu;
        }

        foreach ($items as $key => $item) {
                switch ($item->getName()) {
                        case 'likes':
                                // Remove the "likes" menu item
                                unset($items[$key]);
                                break;
                        case 'edit':
                                // Change the "Edit" text into a custom icon
                                $item->setText(elgg_view_icon('pencil'));
                                break;
                }
        }

        return $items;
}
```

3.15.3 Creating a new menu

Elgg provides multiple different menus by default. Sometimes you may however need some menu items that don't fit in any of the existing menus. If this is the case, you can create your very own menu with the elgg_view_menu() function. You must call the function from the view, where you want to menu to be displayed.

Example: Display a menu called «my_menu» that displays it's menu items in alphapetical order:

```
// in a resource view
echo elgg_view_menu('my_menu', array('sort_by' => 'title'));
```

You can now add new items to the menu like this:

```
// in plugin init
elgg_register_menu_item('my_menu', array(
```

(continué en la próxima página)

(proviene de la página anterior)

```
        'name' => 'my_page',
        'href' => 'path/to/my_page',
        'text' => elgg_echo('my_plugin:my_page'),
));
```

Furthermore it is now possible to modify the menu using the hooks `'register', 'menu:my_menu'` and `'prepare', 'menu:my_menu'`.

3.15.4 Theming

The menu name, section names, and item names are all embedded into the HTML as CSS classes (normalized to contain only hyphens, rather that underscores or colons). This increases the size of the markup slightly but provides themers with a high degree of control and flexibility when styling the site.

Example: The following would be the output of the `foo` menu with sections `alt` and `default` containing items `baz` and `bar` respectively.

```
<ul class="elgg-menu elgg-menu-foo elgg-menu-foo-alt">
        <li class="elgg-menu-item elgg-menu-item-baz"></li>
</ul>
<ul class="elgg-menu elgg-menu-foo elgg-menu-foo-default">
        <li class="elgg-menu-item elgg-menu-item-bar"></li>
</ul>
```

3.15.5 JavaScript

It is common that menu items rely on JavaScript. You can bind client-side events to menu items by placing your JavaScript into AMD module and defining the requirement during the registration.

```
elgg_register_menu_item('my_menu', array(
        'name' => 'hide_on_click',
        'href' => '#',
        'text' => elgg_echo('hide:on:click'),
        'item_class' => '.hide-on-click',
        'deps' => ['navigation/menu/item/hide_on_click'],
));
```

```
// in navigation/menu/item/hide_on_click.js
define(function(require) {
    var $ = require('jquery');

    $(document).on('click', '.hide-on-click', function(e) {
        e.preventDefault();
        $(this).hide();
    });
});
```

3.16 Notificaciones

There are two ways to send notifications in Elgg:

- Instant notifications

- Event-based notifications send using a notifications queue

Contents

3.16.1 Instant notifications

The generic method to send a notification to a user is via the function notify_user(). It is normally used when we want to notify only a single user. Notification like this might for example inform that someone has liked or commented the user's post.

The function usually gets called in an *action* file.

Ejemplo:

In this example a user (`$user`) is triggering an action to rate a post created by another user (`$owner`). After saving the rating (`ElggAnnotation $rating`) to database, we could use the following code to send a notification about the new rating to the owner.

```
// Subject of the notification
$subject = elgg_echo('ratings:notification:subject', array(), $owner->language);

// Summary of the notification
$summary = elgg_echo('ratings:notification:summary', array($user->name), $owner->
↪language);

// Body of the notification message
$body = elgg_echo('ratings:notification:body', array(
        $user->name,
        $owner->name,
        $rating->getValue() // A value between 1-5
), $owner->language);

$params = array(
        'object' => $rating,
        'action' => 'create',
        'summary' => $summary
);

// Send the notification
notify_user($owner->guid, $user->guid, $subject, $body, $params);
```

Nota: The language used by the recipient isn't necessarily the same as the language of the person who triggers the notification. Therefore you must always remember to pass the recipient's language as the third parameter to `elgg_echo()`.

Nota: The `'summary'` parameter is meant for notification plugins that only want to display a short message instead of both the subject and the body. Therefore the summary should be terse but still contain all necessary information.

3.16.2 Enqueued notifications

On large sites there may be many users who have subscribed to receive notifications about a particular event. Sending notifications immediately when a user triggers such an event might remarkably slow down page loading speed. This is why sending of such notifications shoud be left for Elgg's notification queue.

New notification events can be registered with the elgg_register_notification_event() function. Notifications about registered events will be sent automatically to all subscribed users.

This is the workflow of the notifications system:

1. **Someone does an action that triggers an event within Elgg**
    - The action can be `create`, `update` or `delete`
    - The target of the action can be any instance of the `ElggEntity` class (e.g. a Blog post)
2. The notifications system saves this event into a notifications queue in the database
3. When the pluging hook handler for the one-minute interval gets triggered, the event is taken from the queue and it gets processed
4. **Subscriptions are fetched for the user who triggered the event**
    - By default this includes all the users who have enabled any notification method for the user at `www.site.com/notifications/personal/<username>`
5. Plugins are allowed to alter the subscriptions using the `[get, subscriptions]` hook
6. Plugins are allowed to terminate notifications queue processing with the `[send:before, notifications]` hook
7. Plugins are allowed to alter the notification parameters with the `[prepare, notification]` hook
8. Plugins are allowed to alter the notification subject/message/summary with the `[prepare, notification:<action>:<type>:<subtype>]` hook
9. Plugins are allowed to format notification subject/message/summary for individual delivery methods with `[format, notification:<method>]` hook
10. **Notifications are sent to each subscriber using the methods they have chosen**
    - Plugins can take over or prevent sending of each individual notification with the `[send, notification:<method>]` hook
11. The `[send:after, notifications]` hook is triggered for the event after all notifications have been sent

Example

Tell Elgg to send notifications when a new object of subtype «photo» is created:

```
/**
 * Initialize the photos plugin
 */
function photos_init() {
```

(continué en la próxima página)

(proviene de la página anterior)

```
        elgg_register_notification_event('object', 'photo', array('create'));
}
```

Nota: In order to send the event-based notifications you must have the one-minute *CRON* interval configured.

Contents of the notification message can be defined with the `'prepare', 'notification:[action]:[type]:[subtype]'` hook.

Example

Tell Elgg to use the function `photos_prepare_notification()` to format the contents of the notification when a new objects of subtype "photo" is created:

```
/**
 * Initialize the photos plugin
 */
function photos_init() {
    elgg_register_notification_event('object', 'photo', array('create'));
    elgg_register_plugin_hook_handler('prepare', 'notification:create:object:photo',
↪'photos_prepare_notification');
}

/**
 * Prepare a notification message about a new photo
 *
 * @param string                           $hook         Hook name
 * @param string                           $type         Hook type
 * @param Elgg_Notifications_Notification $notification The notification to prepare
 * @param array                            $params       Hook parameters
 * @return Elgg_Notifications_Notification
 */
function photos_prepare_notification($hook, $type, $notification, $params) {
    $entity = $params['event']->getObject();
    $owner = $params['event']->getActor();
    $recipient = $params['recipient'];
    $language = $params['language'];
    $method = $params['method'];

    // Title for the notification
    $notification->subject = elgg_echo('photos:notify:subject', array($entity->title),
↪ $language);

    // Message body for the notification
    $notification->body = elgg_echo('photos:notify:body', array(
        $owner->name,
        $entity->title,
        $entity->getExcerpt(),
        $entity->getURL()
    ), $language);

    // Short summary about the notification
    $notification->summary = elgg_echo('photos:notify:summary', array($entity->title),
↪ $language);
```

(continué en la próxima página)

(proviene de la página anterior)

```
    return $notification;
}
```

Nota: Make sure the notification will be in the correct language by passing the reciepient's language into the `elgg_echo()` function.

3.16.3 Registering a new notification method

By default Elgg has two notification methods: email and the bundled site_notifications plugin. You can register a new notification method with the elgg_register_notification_method() function.

Ejemplo:

Register a handler that will send the notifications via SMS.

```
/**
 * Initialize the plugin
 */
function sms_notifications_init () {
        elgg_register_notification_method('sms');
}
```

After registering the new method, it will appear to the notification settings page at `www.example.com/notifications/personal/[username]`.

3.16.4 Sending the notifications using your own method

Besides registering the notification method, you also need to register a handler that takes care of actually sending the SMS notifications. This happens with the `'send', 'notification:[method]'` hook.

Ejemplo:

```
/**
 * Initialize the plugin
 */
function sms_notifications_init () {
        elgg_register_notification_method('sms');
        elgg_register_plugin_hook_handler('send', 'notification:sms', 'sms_
↪notifications_send');
}

/**
 * Send an SMS notification
 *
 * @param string $hook   Hook name
 * @param string $type   Hook type
 * @param bool   $result Has anyone sent a message yet?
 * @param array  $params Hook parameters
```

(continué en la próxima página)

(proviene de la página anterior)

```
 * @return bool
 * @access private
 */
function sms_notifications_send($hook, $type, $result, $params) {
        /* @var Elgg_Notifications_Notification $message */
        $message = $params['notification'];

        $recipient = $message->getRecipient();

        if (!$recipient || !$recipient->mobile) {
                return false;
        }

        // (A pseudo SMS API class)
        $sms = new SmsApi();

        return $sms->send($recipient->mobile, $message->body);
}
```

3.16.5 Subscriptions

In most cases Elgg core takes care of handling the subscriptions, so notification plugins don't usually have to alter them.

Subscriptions can however be:

- Added using the elgg_add_subscription() function
- Removed using the elgg_remove_subscription() function

It's possible to modify the recipients of a notification dynamically with the `'get', 'subscriptions'` hook.

Ejemplo:

```
/**
 * Initialize the plugin
 */
function discussion_init() {
        elgg_register_plugin_hook_handler('get', 'subscriptions', 'discussion_get_
↪subscriptions');
}

/**
 * Get subscriptions for group notifications
 *
 * @param string $hook          'get'
 * @param string $type          'subscriptions'
 * @param array  $subscriptions Array containing subscriptions in the form
 *                      <user guid> => array('email', 'site', etc.)
 * @param array  $params        Hook parameters
 * @return array
 */
function discussion_get_subscriptions($hook, $type, $subscriptions, $params) {
        $reply = $params['event']->getObject();
```

(continué en la próxima página)

(proviene de la página anterior)

```
        if (!elgg_instanceof($reply, 'object', 'discussion_reply',
↪'ElggDiscussionReply')) {
                return $subscriptions;
        }

        $group_guid = $reply->getContainerEntity()->container_guid;
        $group_subscribers = elgg_get_subscriptions_for_container($group_guid);

        return ($subscriptions + $group_subscribers);
}
```

3.17 Page handler

Elgg offers a facility to manage your plugin pages via a page handler, enabling custom urls like `http://yoursite/your_plugin/section`. To add a page handler to a plugin, a handler function needs to be registered in the plugin's `start.php` file with `elgg_register_page_handler()`:

```
elgg_register_page_handler('your_plugin', 'your_plugin_page_handler');
```

The plugin's page handler is passed two parameters:

- an array containing the sections of the URL exploded by "/". With this information the handler will be able to apply any logic necessary, for example loading the appropriate view and returning its contents.
- the handler, this is the handler that is currently used (in our example `your_plugin`). If you don't register multiple page handlers to the same function you'll never need this.

3.17.1 Code flow

Pages in plugins should be rendered via page handlers (not by using `Elgg\Application`). Generally the rendering is done by views with names starting with `resources/`. The program flow is something like this:

1. A user requests `/plugin_name/section/entity`
2. Elgg checks if `plugin_name` is registered to a page handler and calls that function, passing `array('section', 'entity')` as the first argument
3. The page handler function determines which resource view will display the page.
4. The handler uses `elgg_view_resource()` to render the page, also passing in any relevant info to the view via the `$vars` argument.
5. The resource view combines many separate views, calls formatting functions like `elgg_view_layout()` and `elgg_view_page()`, and then echos the final output
6. The user sees a fully rendered page

There is no syntax enforced on the URLs, but Elgg's coding standards suggests a certain format.

3.18 Encaminamiento

Elgg has two mechanisms to respond to HTTP requests that don't already go through the *Actions* and *Simplecache* systems.

3.18.1 URL Identifier and Segments

After removing the site URL, Elgg splits the URL path by `/` into an array. The first element, the **identifier**, is shifted off, and the remaining elements are called the **segments**. For example, if the site URL is `http://example.com/elgg/`, the URL `http://example.com/elgg/blog/owner/jane?foo=123` produces:

Identifier: `'blog'`. Segments: `['owner', 'jane']`. (the query string parameters are available via `get_input()`)

The site URL (home page) is a special case that produces an empty string identifier and an empty segments array.

> **Advertencia:** URL identifier/segments should be considered potentially dangerous user input. Elgg uses `htmlspecialchars` to escapes HTML entities in them.

3.18.2 Page Handler

To handle all URLs that begin with a particular identifier, you can register a function to act as a *Page handler*. When the handler is called, the segments array is passed in as the first argument.

The following code registers a page handler for «blog» URLs and shows how one might route the request to a resource view.

```
elgg_register_page_handler('blog', 'blog_page_handler');

function blog_page_handler(array $segments) {
     // if the URL is http://example.com/elgg/blog/view/123/my-blog-post
     // $segments contains: ['view', '123', 'my-blog-post']

     $subpage = elgg_extract(0, $segments);
     if ($subpage === 'view') {

          // use a view for the page logic to allow other plugins to easily change it
          $resource = elgg_view_resource('blog/view', [
              'guid' => (int)elgg_extract(1, $segments);
          ]);

          return elgg_ok_response($resource);
     }

     // redirect to a different location
     if ($subpage === '') {
          return elgg_redirect_response('blog/all');
     }

     // send an error page
     if ($subpage === 'owner' && !elgg_entity_exists($segments[1])) {
          return elgg_error_response('User not found', 'blog/all', ELGG_HTTP_NOT_
↪FOUND);
     }

     // ... handle other subpages
}
```

3.18.3 The `route` Plugin Hook

The `route` plugin hook is triggered before page handlers are called. The URL identifier is given as the type of the hook. This hook can be used to add some logic before the request is handled elsewhere, or take over page rendering completely.

Generally devs should instead use a page handler unless they need to affect a single page or a wider variety of URLs.

The following code results in `/blog/all` requests being completely handled by the plugin hook handler. For these requests the `blog` page handler is never called.

```
function myplugin_blog_all_handler($hook, $type, $returnvalue, $params) {
    $segments = elgg_extract('segments', $returnvalue, array());

    if (isset($segments[0]) && $segments[0] === 'all') {
        $title = "We're taking over!";
        $content = elgg_view_layout('one_column', array(
            'title' => $title,
            'content' => "We can take over page rendering completely"
        ));
        echo elgg_view_page($title, $content);

        // in the route hook, return false says, "stop rendering, we've handled this␣
↪request"
        return false;
    }
}

elgg_register_plugin_hook_handler('route', 'blog', 'myplugin_blog_all_handler');
```

Nota: As of 2.1, route modification should be done in the `route:rewrite` hook.

3.18.4 The `route:rewrite` Plugin Hook

For URL rewriting, the `route:rewrite` hook (with similar arguments as `route`) is triggered very early, and allows modifying the request URL path (relative to the Elgg site).

Here we rewrite requests for `news/*` to `blog/*`:

```
function myplugin_rewrite_handler($hook, $type, $value, $params) {
    $value['identifier'] = 'blog';
    return $value;
}

elgg_register_plugin_hook_handler('route:rewrite', 'news', 'myplugin_rewrite_handler
↪');
```

Advertencia: The hook must be registered directly in your plugin `start.php` (the `[init, system]` event is too late).

3.18.5 Routing overview

For regular pages, Elgg's program flow is something like this:

1. A user requests `http://example.com/news/owner/jane`.
2. Plugins are initialized.
3. Elgg parses the URL to identifier `news` and segments `['owner', 'jane']`.
4. Elgg triggers the plugin hook `route:rewrite, news` (see above).
5. Elgg triggers the plugin hook `route, blog` (was rewritten in the rewrite hook).
6. Elgg finds a registered page handler (see above) for `blog`, and calls the function, passing in the segments.
7. The page handler function determines it needs to render a single user's blog. It calls `elgg_view_resource('blog/owner', $vars)` where `$vars` contains the username.
8. The `resources/blog/owner` view gets the username via `$vars['username']`, and uses many other views and formatting functions like `elgg_view_layout()` and `elgg_view_page()` to create the entire HTML page.
9. The page handler echos the view HTML and returns `true` to indicate it handled the request.
10. PHP invokes Elgg's shutdown sequence.
11. The user receives a fully rendered page.

Elgg's coding standards suggest a particular URL layout, but there is no syntax enforced.

3.19 Services

Elgg uses the `Elgg\Application` class to load and bootstrap Elgg. In future releases this class will offer a set of service objects for plugins to use.

Nota: If you have a useful idea, you can *add a new service*!

3.19.1 Menus

`elgg()->menus` provides low-level methods for constructing menus. In general, menus should be passed to `elgg_view_menu` for rendering instead of manual rendering.

3.20 Page ownership

One recurring task of any plugin will be to determine the page ownership in order to decide which actions are allowed or not. Elgg has a number of functions related to page ownership and also offers plugin developers flexibility by letting the plugin handle page ownership requests as well. Determining the owner of a page can be determined with `elgg_get_page_owner_guid()`, which will return the GUID of the owner. Alternatively, `elgg_get_page_owner_entity()` will retrieve the whole page owner entity. If the page already knows who the page owner is, but the system doesn't, the page can set the page owner by passing the GUID to `elgg_set_page_owner_guid($guid)`.

Nota: The page owner entity can be any `ElggEntity`. If you wish to only apply some setting in case of a user or a group make sure you check that you have the correct entity.

3.20.1 Custom page owner handlers

Plugin developers can create page owner handlers, which could be necessary in certain cases, for example when integrating third party functionality. The handler will be a function which will need to get registered with `elgg_register_plugin_hook_handler('page_owner', 'system', 'your_page_owner_function_name');` . The handler will only need to return a value (an integer GUID) when it knows for certain who the page owner is.

By default, the system uses `default_page_owner_handler()` to determine the page_owner from the following elements:

- The `username` URL parameter
- The `owner_guid` URL parameter
- The URL path

It then passes off to any page owner handlers defined using the *plugin hook*. If no page owner can be determined, the page owner is set to 0, which is the same as the logged out user.

3.21 Permissions Check

Advertencia: As stated in the page, this method works **only** for granting **write** access to entities. You **cannot** use this method to retrieve or view entities for which the user does not have read access.

Elgg provides a mechanism of overriding write permissions check through the *permissions_check plugin hook* . This is useful for allowing plugin write to all accessible entities regardless of access settings. Entities that are hidden, however, will still be unavailable to the plugin.

3.21.1 Hooking permissions_check

In your plugin, you must register the plugin hook for `permissions_check`.

```
elgg_register_plugin_hook_handler('permissions_check', 'all', 'myplugin_permissions_
↪check');
```

3.21.2 The override function

Now create the function that will be called by the permissions check hook. In this function we determine if the entity (in parameters) has write access. Since it is important to keep Elgg secure, write access should be given only after checking a variety of situations including page context, logged in user, etc. Note that this function can return 3 values: true if the entity has write access, false if the entity does not, and null if this plugin doesn't care and the security system should consult other plugins.

```
function myplugin_permissions_check($hook_name, $entity_type, $return_value,
↪$parameters) {
   $has_access = determine_access_somehow();

   if ($has_access === true) {
      return true;
   } else if ($has_access === false) {
      return false;
   }

   return null;
}
```

3.21.3 Full Example

This is a full example using the context to determine if the entity has write access.

```
<?php

function myaccess_init() {
   // Register cron hook
   if (!elgg_get_plugin_setting('period', 'myaccess')) {
      elgg_set_plugin_setting('period', 'fiveminute', 'myaccess');
   }

   // override permissions for the myaccess context
   elgg_register_plugin_hook_handler('permissions_check', 'all', 'myaccess_
↪permissions_check');

   elgg_register_plugin_hook_handler('cron', elgg_get_plugin_setting('period',
↪'myaccess'), 'myaccess_cron');
}

/**
 * Hook for cron event.
 */
function myaccess_cron($event, $object_type, $object) {

   elgg_push_context('myaccess_cron');

   // returns all entities regardless of access permissions.
   // will NOT return hidden entities.
   $entities = get_entities();

   elgg_pop_context();
}

/**
 * Overrides default permissions for the myaccess context
 */
function myaccess_permissions_check($hook_name, $entity_type, $return_value,
↪$parameters) {
   if (elgg_in_context('myaccess_cron')) {
      return true;
   }
```

(continué en la próxima página)

(proviene de la página anterior)

```
   return null;
}

// Initialise plugin
register_elgg_event_handler('init', 'system', 'myaccess_init');
?>
```

3.22 Plugin settings

You need to perform some extra steps if your plugin needs settings to be saved and controlled via the administration panel:

- Create a file in your plugin's default view folder called `plugins/your_plugin/settings.php`, where `your_plugin` is the name of your plugin's directory in the `mod` hierarchy
- Fill this file with the form elements you want to display together with *internationalised* text labels
- Set the name attribute in your form components to `param[`varname`]` where `varname` is the name of the variable. These will be saved as private settings attached to a plugin entity. So, if your variable is called `param[myparameter]` your plugin (which is also passed to this view as `$vars['entity']`) will be called `$vars['entity']->myparameter`

An example `settings.php` would look like:

```
<p>
   <?php echo elgg_echo('myplugin:settings:limit'); ?>

   <select name="params[limit]">
      <option value="5" <?php if ($vars['entity']->limit == 5) echo " selected=\"yes\
↪" "; ?>>5</option>
      <option value="8" <?php if ((!$vars['entity']->limit) || ($vars['entity']->
↪limit == 8)) echo " selected=\"yes\" "; ?>>8</option>
      <option value="12" <?php if ($vars['entity']->limit == 12) echo " selected=\
↪"yes\" "; ?>>12</option>
      <option value="15" <?php if ($vars['entity']->limit == 15) echo " selected=\
↪"yes\" "; ?>>15</option>
   </select>
</p>
```

Nota: You don't need to add a save button or the form, this will be handled by the framework.

Nota: You cannot use form components that send no value when «off.» These include radio inputs and check boxes.

3.22.1 User settings

Your plugin might need to store per user settings too, and you would like to have your plugin's options to appear in the user's settings page. This is also easy to do and follows the same pattern as setting up the global plugin configuration explained earlier. The only difference is that instead of using a `settings` file you will use `usersettings`. So, the path to the user edit view for your plugin would be `plugins/your_plugin/usersettings.php`.

Nota: The title of the usersettings form will default to the plugin name. If you want to change this, add a translation for `plugin_id:usersettings:title`.

3.22.2 Retrieving settings in your code

To retrieve settings from your code use:

```
$setting = elgg_get_plugin_setting($name, $plugin_id);
```

or for user settings

```
$user_setting = elgg_get_plugin_user_setting($name, $user_guid, $plugin_id);
```

where:

- `$name` Is the value you want to retrieve
- `$user_guid` Is the user you want to retrieve these for (defaults to the currently logged in user)
- `$plugin_name` Is the name of the plugin (detected if run from within a plugin)

3.22.3 Setting values while in code

Values may also be set from within your plugin code, to do this use one of the following functions:

```
elgg_set_plugin_setting($name, $value, $plugin_id);
```

or

```
elgg_set_plugin_user_setting($name, $value, $user_guid, $plugin_id);
```

Advertencia: The `$plugin_id` needs to be provided when setting plugin (user)settings.

3.23 River

Elgg natively supports the «river», an activity stream containing descriptions of activities performed by site members. This page gives an overview of adding events to the river in an Elgg plugin.

3.23.1 Pushing river items

Items are pushed to the activity river through a function call, which you must include in your plugins for the items to appear.

Here we add a river item telling that a user has created a new blog post:

```
<?php

elgg_create_river_item(array(
```

(continué en la próxima página)

(proviene de la página anterior)

```
        'view' => 'river/object/blog/create',
        'action_type' => 'create',
        'subject_guid' => $blog->owner_guid,
        'object_guid' => $blog->getGUID(),
));
```

All available parameters:

- `view` => STR The view that will handle the river item (must exist)
- `action_type` => STR An arbitrary string to define the action (e.g. “create”, “update”, “vote”, “review”, etc)
- `subject_guid` => INT The GUID of the entity doing the action
- `object_guid` => INT The GUID of the entity being acted upon
- `target_guid` => INT The GUID of the the object entity’s container (optional)
- `access_id` => INT The access ID of the river item (default: same as the object)
- `posted` => INT The UNIX epoch timestamp of the river item (default: now)
- `annotation_id` => INT The annotation ID associated with this river entry (optional)

When an item is deleted or changed, the river item will be updated automatically.

3.23.2 River views

In order for events to appear in the river you need to provide a corresponding *view* with the name specified in the function above.

We recommend `/river/{type}/{subtype}/{action}`, where:

- {type} is the entity type of the content we’re interested in (`object` for objects, `user` for users, etc)
- {subtype} is the entity subtype of the content we’re interested in (`blog` for blogs, `photo_album` for albums, etc)
- {action} is the action that took place (“”create””, “”update””, etc)

River item information will be passed in an object called `$vars['item']`, which contains the following important parameters:

- `$vars['item']->subject_guid` The GUID of the user performing the action
- `$vars['item']->object_guid` The GUID of the entity being acted upon

Timestamps etc will be generated for you.

For example, the blog plugin uses the following code for its river view:

```
<?php

$object = $vars['item']->getObjectEntity();

$excerpt = $object->excerpt ? $object->excerpt : $object->description;
$excerpt = strip_tags($excerpt);
$excerpt = elgg_get_excerpt($excerpt);

echo elgg_view('river/elements/layout', array(
        'item' => $vars['item'],
```

(continué en la próxima página)

(proviene de la página anterior)

```
        'message' => $excerpt,
));
```

3.24 Temas

Personalice el aspecto de Elgg.

Un tema es un tipo de doc:*complemento </admin/plugins>* que substituye los aspectos visibles de Elgg.

Esta guía asume que usted está ya familiarizado con:

- *Complementos*
- *Vistas*

Contents

3.24.1 Cree su complemento

Cree su complemento siguiendo las instrucciones descritas en la *guía para desarrolladores*.

- Cree una carpeta nueva dentro de `mod/`.
- Cree un nuevo `start.php`.
- Cree un nuevo fichero `manifest.xml` que describa el tema.

3.24.2 Personalice el CSS

Desde la versión 1.8 de Elgg, el código CSS está dividido en varios ficheros en base a qué aspectos del sitio son los que cambia el tema. Esto le permite trabajar en el aspecto de los distintos elementos del tema de uno en uno, de forma que pueda progresar rápidamente sin necesidad de angustiarse.

La siguiente es una lista de las vistas de CSS existentes:

- elements/buttons.css: Provides a way to style all the different kinds of buttons your site will use. There are 5 kinds of buttons that plugins will expect to be available: action, cancel, delete, submit, and special.
- elements/chrome.css: This file has some miscellaneous look-and-feel classes.
- elements/components.css: This file contains many "css objects" that are used all over the site: media block, list, gallery, table, owner block, system messages, river, tags, photo, and comments.

- elements/forms.css: This file determines what your forms and input elements will look like.
- elements/icons.css: Contains styles for the icons and avatars used on your site.
- elements/layout.css: Determines what your page layout will look like: sidebars, page wrapper, main body, header, footer, etc.
- elements/modules.css: Lots of content in Elgg is displayed in boxes with a title and a content body. We called these modules. There are a few kinds: info, aside, featured, dropdown, popup, widget. Widget styles are included in this file too, since they are a subset of modules.
- elements/navigation.css: This file determines what all your menus will look like.
- elements/typography.css: This file determines what the content and headings of your site will look like.
- rtl.css: Custom rules for users viewing your site in a right-to-left language.
- admin.css: A completely separate theme for the admin area (usually not overridden).
- elgg.css: Compiles all the core elements/* files into one file (DO NOT OVERRIDE).
- elements/core.css: Contains base styles for the more complicated "css objects". If you find yourself wanting to override this, you probably need to report a bug to Elgg core instead (DO NOT OVERRIDE).
- elements/reset.css: Contains a reset stylesheet that forces elements to have the same default

Extensión de vistas

Existen dos maneras de modificar vistas:

La primera forma consiste en añadir cosas adicionales a una vista existente mediante la función de extender la vista desde dentro de la función de inicio de `start.php`.

Por ejemplo, el siguiente `start.php` añadirá `mytheme/css` al fichero CSS del núcleo de Elgg:

```
<?php

    function mytheme_init() {
        elgg_extend_view('elgg.css', 'mytheme/css');
    }

    elgg_register_event_handler('init', 'system', 'mytheme_init');
?>
```

Substitución de vistas

Los complementos pueden tener una jerarquía de vistas, todo fichero que exista aquí substituirá a cualquier fichero en la jerarquía de vistas del núcleo de Elgg. Por ejemplo, si el complemento tiene un fichero:

`/mod/myplugin/views/default/elements/typography.css`

Este fichero substituirá a:

`/views/default/elements/typography.css`

Pero sólo mientras el complemento esté activado.

This gives you total control over the way Elgg looks and behaves. It gives you the option to either slightly modify or totally replace existing views.

Icons

As of Elgg 2.0 the default Elgg icons come from the FontAwesome library. You can use any of these icons by calling:

```
elgg_view_icon('icon-name');
```

`icon-name` can be any of the FontAwesome icons without the `fa-`-prefix.

3.24.3 Herramientas

Desde la versión 1.8 de Elgg, se ofrecen algunas herramientas de desarrollo para ayudar a desarrollar temas. Active el complemento de «Desarrolladores» y acceda a la página «Vista previa del tema» para observar el progreso del tema.

3.24.4 Personalizar la página principal

La página principal de Elgg ejecuta un gancho de complementos llamado `index,system`. Si el gancho devuelve `true`, se asume que ya se ha dibujado otra página principal, y no se muestra la página predeterminada.

Por lo tanto, puede substituir la página principal registrando una función para el gancho de complementos `index, system` y devolviendo `true` desde esa función.

Resumiendo:

- Cree un complemento nuevo.
- Necesitará algo como lo siguiente en `start.php`:

```
<?php

function pluginname_init() {
    // Replace the default index page
    elgg_register_plugin_hook_handler('index', 'system', 'new_index');
}

function new_index() {
    if (!include_once(dirname(dirname(__FILE__)) . "/pluginname/pages/index.php"))
        return false;

    return true;
}

// register for the init, system event when our plugin start.php is loaded
elgg_register_event_handler('init', 'system', 'pluginname_init');
?>
```

- A continuación, cree la página principal (`/pluginname/pages/index.php`) y úsela para poner el contenido que le gustaría tener en la página principal de su sitio Elgg.

3.25 Vistas

Contents

3.25.1 Introduction

Views are responsible for creating output. They handle everything from:

- the layout of pages
- chunks of presentation output (like a footer or a toolbar)
- individual links and form inputs.
- the images, js, and css needed by your web page

3.25.2 Using views

At their most basic level, the default views are just PHP files with snippets of html:

```
<h1>Hello, World!</h1>
```

Assuming this view is located at `/views/default/hello.php`, we could output it like so:

```
echo elgg_view('hello');
```

For your convenience, Elgg comes with quite a lot of views by default. In order to keep things manageable, they are organized into subdirectories. Elgg handles this situation quite nicely. For example, our simple view might live in `/views/default/hello/world.php`, in which case it would be called like so:

```
echo elgg_view('hello/world');
```

The name of the view simply reflects the location of the view in the views directory.

3.25.3 Views as templates

You can pass arbitrary data to a view via the `$vars` array. Our `hello/world` view might be modified to accept a variable like so:

```
<h1>Hello, <?= $vars['name']; ?>!</h1>
```

In this case, we can pass an arbitrary name parameter to the view like so:

```
echo elgg_view('hello/world', ['name' => 'World']);
```

which would produce the following output:

```
<h1>Hello, World!</h1>
```

Advertencia: Views don't do any kind of automatic output sanitization by default. You are responsible for doing the correct sanitization yourself to prevent XSS attacks and the like.

3.25.4 Views as cacheable assets

As mentioned before, views can contain JS, CSS, or even images.

Asset views must meet certain requirements:

- They *must not* take any `$vars` parameters
- They *must not* change their output based on global state like
  - who is logged in
  - the time of day
- They *must* contain a valid file extension
  - Bad: `my/cool/template`
  - Good: `my/cool/template.html`

For example, suppose you wanted to load some CSS on a page. You could define a view `mystyles.css`, which would look like so:

```
/* /views/default/mystyles.css */
.mystyles-foo {
  background: red;
}
```

Nota: Leave off the trailing «.php» from the filename and Elgg will automatically recognize the view as cacheable.

To get a URL to this file, you would use `elgg_get_simplecache_url`:

```
// Returns "https://mysite.com/.../289124335/default/mystyles.css
elgg_get_simplecache_url('mystyles.css');
```

Elgg automatically adds the magic numbers you see there for cache-busting and sets long-term expires headers on the returned file.

Advertencia: Elgg may decide to change the location or structure of the returned URL in a future release for whatever reason, and the cache-busting numbers change every time you flush Elgg's caches, so the exact URL is not stable by design.

With that in mind, here's a couple anti-patterns to avoid:

- Don't rely on the exact structure/location of this URL

- Don't try to generate the URLs yourself
- Don't store the returned URLs in a database

In your plugin's init function, register the css file:

```
elgg_register_css('mystyles', elgg_get_simplecache_url('mystyles.css'));
```

Then on the page you want to load the css, call:

```
elgg_load_css('mystyles');
```

3.25.5 Views and third-party assets

The best way to serve third-party assets is through views. However, instead of manually copy/pasting the assets into the right location in `/views/*`, you can map the assets into the views system via the `"views"` key in your plugin's `elgg-plugin.php` config file.

The views value must be a 2 dimensional array. The first level maps a viewtype to a list of view mappings. The secondary lists map view names to file paths, either absolute or relative to the Elgg root directory.

If you check your assets into source control, point to them like this:

```
<?php // mod/example/elgg-plugin.php
return [
    // view mappings
    'views' => [
        // viewtype
        'default' => [
            // view => /path/from/filesystem/root
            'js/jquery-ui.js' => __DIR__ . '/bower_components/jquery-ui/jquery-ui.min.
↪js',
        ],
    ],
];
```

To point to assets installed with `fxp/composer-asset-plugin`, use install-root-relative paths by leaving off the leading slash:

```
<?php // mod/example/elgg-plugin.php
return [
    'views' => [
        'default' => [
            // view => path/from/install/root
            'js/jquery-ui.js' => 'vendor/bower-asset/jquery-ui/jquery-ui.min.js',
        ],
    ],
];
```

Elgg core uses this feature extensively, though the value is returned directly from `/engine/views.php`.

Nota: You don't have to use Bower, Composer Asset Plugin, or any other script for managing your plugin's assets, but we highly recommend using a package manager of some kind because it makes upgrading so much easier.

Specifying additional views directories

In `elgg-plugin.php` you can also specify directories to be scanned for views. Just provide a view name prefix ending with `/` and a directory path (like above).

```
<?php // mod/file/elgg-plugin.php
return [
    'views' => [
        'default' => [
            'file/icon/' => __DIR__ . '/graphics/icons',
        ],
    ],
];
```

With the above, files found within the `icons` folder will be interpreted as views. E.g. the view `file/icon/general.gif` will be created and mapped to `mod/file/graphics/icons/general.gif`.

Nota: This is a fully recursive scan. All files found will be brought into the views system.

Multiple paths can share the same prefix, just give an array of paths:

```
<?php // mod/file/elgg-plugin.php
return [
    'views' => [
        'default' => [
            'file/icon/' => [
                __DIR__ . '/graphics/icons',
                __DIR__ . '/more_icons', // processed 2nd (may override)
            ],
        ],
    ],
];
```

3.25.6 Viewtypes

You might be wondering: «Why `/views/default/hello/world.php` instead of just `/views/hello/world.php`?».

The subdirectory under `/views` determines the *viewtype* of the views below it. A viewtype generally corresponds to the output format of the views.

The default viewtype is assumed to be HTML and other static assets necessary to render a responsive web page in a desktop or mobile browser, but it could also be:

- RSS
- ATOM
- JSON
- Mobile-optimized HTML
- TV-optimized HTML
- Any number of other data formats

You can force Elgg to use a particular viewtype to render the page by setting the `view` input variable like so: `https://mysite.com/?view=rss`.

You could also write a plugin to set this automatically using the `elgg_set_viewtype()` function. For example, your plugin might detect that the page was accessed with an iPhone's browser string, and set the viewtype to `iphone` by calling:

```
elgg_set_viewtype('iphone');
```

The plugin would presumably also supply a set of views optimized for those devices.

3.25.7 Altering views via plugins

Without modifying Elgg's core, Elgg provides several ways to customize almost all output:

- You can *override a view*, completely changing the file used to render it.
- You can *extend a view* by prepending or appending the output of another view to it.
- You can *alter a view's inputs* by plugin hook.
- You can *alter a view's output* by plugin hook.

Overriding views

Views in plugin directories always override views in the core directory; however, when plugins override the views of other plugins, *later plugins take precedent*.

For example, if we wanted to customize the `hello/world` view to use an `h2` instead of an `h1`, we could create a file at `/mod/example/views/default/hello/world.php` like this:

```
<h2>Hello, <?= $vars['name']; ?></h2>
```

Nota: When considering long-term maintenance, overriding views in the core and bundled plugins has a cost: Upgrades may bring changes in views, and if you have overridden them, you will not get those changes.

You may instead want to alter *the input* or *the output* of the view via plugin hooks.

Nota: Elgg caches view locations. This means that you should disable the system cache while developing with views. When you install the changes to a production environment you must flush the caches.

Extending views

There may be other situations in which you don't want to override the whole view, you just want to prepend or append some more content to it. In Elgg this is called *extending a view*.

For example, instead of overriding the `hello/world` view, we could extend it like so:

```
elgg_extend_view('hello/world', 'hello/greeting');
```

If the contents of `/views/default/hello/greeting.php` is:

```
<h2>How are you today?</h2>
```

Then every time we call `elgg_view('hello/world');`, we'll get:

```
<h1>Hello, World!</h1>
<h2>How are you today?</h2>
```

You can prepend views by passing a value to the 3rd parameter that is less than 500:

```
// appends 'hello/greeting' to every occurrence of 'hello/world'
elgg_extend_view('hello/world', 'hello/greeting');

// prepends 'hello/greeting' to every occurrence of 'hello/world'
elgg_extend_view('hello/world', 'hello/greeting', 450);
```

All view extensions should be registered in your plugin's `init,system` event handler in `start.php`.

Altering view input

It may be useful to alter a view's `$vars` array before the view is rendered.

Since 1.11, before each view rendering the `$vars` array is filtered by the *plugin hook* `["view_vars", $view_name]`. Each registered handler function is passed these arguments:

- `$hook` - the string `"view_vars"`
- `$view_name` - the view name being rendered (the first argument passed to `elgg_view()`)
- `$returnvalue` - the modified `$vars` array
- `$params` - an array containing:
  - `vars` - the original `$vars` array, unaltered
  - `view` - the view name
  - `viewtype` - The *viewtype* being rendered

Altering view input example

Here we'll alter the default pagination limit for the comments view:

```
elgg_register_plugin_hook_handler('view_vars', 'page/elements/comments', 'myplugin_
↪alter_comments_limit');

function myplugin_alter_comments_limit($hook, $type, $vars, $params) {
    // only 10 comments per page
    $vars['limit'] = elgg_extract('limit', $vars, 10);
    return $vars;
}
```

Altering view output

Sometimes it is preferable to alter the output of a view instead of overriding it.

The output of each view is run through the *plugin hook* `["view", $view_name]` before being returned by `elgg_view()`. Each registered handler function is passed these arguments:

- `$hook` - the string `"view"`
- `$view_name` - the view name being rendered (the first argument passed to `elgg_view()`)

- `$result` - the modified output of the view
- `$params` - an array containing:
  - `viewtype` - The *viewtype* being rendered

To alter the view output, the handler just needs to alter `$returnvalue` and return a new string.

Altering view output example

Here we'll eliminate breadcrumbs that don't have at least one link.

```
elgg_register_plugin_hook_handler('view', 'navigation/breadcrumbs', 'myplugin_alter_
↪breadcrumb');

function myplugin_alter_breadcrumb($hook, $type, $returnvalue, $params) {
    // we only want to alter when viewtype is "default"
    if ($params['viewtype'] !== 'default') {
        return $returnvalue;
    }

    // output nothing if the content doesn't have a single link
    if (false === strpos($returnvalue, '<a ')) {
        return '';
    }

    // returning nothing means "don't alter the returnvalue"
}
```

Replacing view output completely

You can pre-set the view output by setting `$vars['__view_output']`. The value will be returned as a string. View extensions will not be used and the `view` hook will not be triggered.

```
elgg_register_plugin_hook_handler('view_vars', 'navigation/breadcrumbs', 'myplugin_no_
↪page_breadcrumbs');

function myplugin_no_page_breadcrumbs($hook, $type, $vars, $params) {
    if (elgg_in_context('pages')) {
        return ['__view_output' => ""];
    }
}
```

3.25.8 Mostrar entidades

If you don't know what an entity is, *check this page out first*.

The following code will automatically display the entity in `$entity`:

```
echo elgg_view_entity($entity);
```

As you'll know from the data model introduction, all entities have a *type* (object, site, user or group), and optionally a subtype (which could be anything - "blog", "forumpost", "banana").

`elgg_view_entity` will automatically look for a view called `type/subtype`; if there's no subtype, it will look for `type/type`. Failing that, before it gives up completely it tries `type/default`.

RSS feeds in Elgg generally work by outputting the `object/default` view in the "rss" viewtype.

For example, the view to display a blog post might be `object/blog`. The view to display a user is `user/default`.

3.25.9 Full and partial entity views

`elgg_view_entity` actually has a number of parameters, although only the very first one is required. The first three are:

- `$entity` - The entity to display
- `$viewtype` - The viewtype to display in (defaults to the one we're currently in, but it can be forced - eg to display a snippet of RSS within an HTML page)
- `$full_view` - Whether to display a *full* version of the entity. (Defaults to false.)

This last parameter is passed to the view as `$vars['full_view']`. It's up to you what you do with it; the usual behaviour is to only display comments and similar information if this is set to true.

3.25.10 Listing entities

This is then used in the provided listing functions. To automatically display a list of blog posts (*see the full tutorial*), you can call:

```
echo elgg_list_entities([
    'type' => 'object',
    'subtype' => 'blog',
]);
```

This function checks to see if there are any entities; if there are, it first displays the `navigation/pagination` view in order to display a way to move from page to page. It then repeatedly calls `elgg_view_entity` on each entity before returning the result.

Note that `elgg_list_entities` allows the URL to set its `limit` and `offset` options, so set those explicitly if you need particular values (e.g. if you're not using it for pagination).

Elgg knows that it can automatically supply an RSS feed on pages that use `elgg_list_entities`. It initializes the `["head","page"]` plugin hook (which is used by the header) in order to provide RSS autodiscovery, which is why you can see the orange RSS icon on those pages in some browsers.

If your entity list will display the entity owners, you can improve performance a bit by preloading all owner entities:

```
echo elgg_list_entities([
    'type' => 'object',
    'subtype' => 'blog',

    // enable owner preloading
    'preload_owners' => true,
]);
```

See also *this background information on Elgg's database*.

If you want to show a message when the list does not contain items to list, you can pass a `no_results` message. If you want even more controle over the `no_results` message you can also pass a Closure (an anonymous function).

```
echo elgg_list_entities([
    'type' => 'object',
    'subtype' => 'blog',
```

(continué en la próxima página)

(proviene de la página anterior)

```
    'no_results' => elgg_echo('notfound'),
]);
```

Rendering a list with an alternate view

Since 1.11, you can define an alternative view to render list items using `'item_view'` parameter.

In some cases, default entity views may be unsuitable for your needs. Using `item_view` allows you to customize the look, while preserving pagination, list's HTML markup etc.

Consider these two examples:

```
echo elgg_list_entities_from_relationship([
    'type' => 'group',
    'relationship' => 'member',
    'relationship_guid' => elgg_get_logged_in_user_guid(),
    'inverse_relationship' => false,
    'full_view' => false,
]);
```

```
echo elgg_list_entities_from_relationship([
    'type' => 'group',
    'relationship' => 'invited',
    'relationship_guid' => (int) $user_guid,
    'inverse_relationship' => true,
    'item_view' => 'group/format/invitationrequest',
]);
```

In the first example, we are displaying a list of groups a user is a member of using the default group view. In the second example, we want to display a list of groups the user was invited to.

Since invitations are not entities, they do not have their own views and can not be listed using `elgg_list_*`. We are providing an alternative item view, that will use the group entity to display an invitation that contains a group name and buttons to access or reject the invitation.

Rendering a list as a table

Since 2.3 you can render lists as tables. Set `$options['list_type'] = 'table'` and provide an array of TableColumn objects as `$options['columns']`. The service `elgg()->table_columns` provides several methods to create column objects based around existing views (like `page/components/column/*`), properties, or methods.

In this example, we list the latest `my_plugin` objects in a table of 3 columns: entity icon, the display name, and a friendly format of the time.

```
echo elgg_list_entities([
    'type' => 'object',
    'subtype' => 'my_plugin',

    'list_type' => 'table',
    'columns' => [
        elgg()->table_columns->icon(),
        elgg()->table_columns->getDisplayName(),
```

(continué en la próxima página)

(proviene de la página anterior)

```
        elgg()->table_columns->time_created(null, [
            'format' => 'friendly',
        ]),
    ],
]);
```

See the `Elgg\Views\TableColumn\ColumnFactory` class for more details on how columns are specified and rendered. You can add or override methods of `elgg()->table_columns` in a variety of ways, based on views, properties/methods on the items, or given functions.

3.25.11 Related

Page structure best practice

Elgg pages have an overall pageshell and a main content area. In Elgg 1.0+, we've marked out a space «the canvas» for items to write to the page. This means the user always has a very consistent experience, while giving maximum flexibility to plugin authors for laying out their functionality.

Think of the canvas area as a big rectangle that you can do what you like in. We've created a couple of standard canvases for you:

- one column
- two column
- content
- widgets

are the main ones. You can access these with the function:

```
$canvas_area = elgg_view_layout($canvas_name, array(
  'content' => $content,
  'section' => $section
));
```

The content sections are passed as an `array` in the second parameter. The array keys correspond to sections in the layout, the choice of layout will determine which sections to pass. The array values contain the html that should be displayed in those areas. Examples of two common layouts:

```
$canvas_area = elgg_view_layout('one_column', array(
  'content' => $content
));
```

```
$canvas_area = elgg_view_layout('one_sidebar', array(
  'content' => $content,
  'sidebar' => $sidebar
));
```

You can then, ultimately, pass this into the `elgg_view_page` function:

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

You may also have noticed that we've started including a standard title area at the top of each plugin page (or at least, most plugin pages). This is created using the following wrapper function, and should usually be included at the top of the plugin content:

```
$start_of_plugin_content = elgg_view_title($title_text);
```

This will also display any submenu items that exist (unless you set the second, optional parameter to false). So how do you add submenu items?

In your plugin_init function, include the following call:

```
if (elgg_get_context() == "your_plugin") {
   // add a site navigation item
   $item = new ElggMenuItem('identifier', elgg_echo('your_plugin:link'), $url);
   elgg_register_menu_item('page', $item);
}
```

The submenu will then automatically display when your page is rendered. The "identifier" is a machine name for the link, it should be unique per menu.

Simplecache

Ver también:

- *Rendimiento*
- *Vistas*

The Simplecache is a mechanism designed to alleviate the need for certain views to be regenerated dynamically. Instead, they are generated once, saved as a static file, and served in a way that entirely bypasses the Elgg engine.

If Simplecache is turned off (which can be done from the administration panel), these views will be served as normal, with the exception of site CSS.

The criteria for whether a view is suitable for the Simplecache is as follows:

- The view must not change depending on who or when it is being looked at
- The view must not depend on variables fed to it (except for global variables like site URL that never change)

Regenerating the Simplecache

You can regenerate the Simplecache at any time by:

- Loading `/upgrade.php`, even if you have nothing to upgrade
- In the admin panel click on "Flush the caches"
- Enabling or disabling a plugin
- Reordering your plugins

Using the Simplecache in your plugins

Registering views with the Simplecache

You can register a view with the Simplecache with the following function at init-time:

```
elgg_register_simplecache_view($viewname);
```

Accessing the cached view

If you registered a JavaScript or CSS file with Simplecache and put in the view folder as `your_view.js` or `your_view.css` you can very easily get the url to this cached view by calling `elgg_get_simplecache_url($view)`. For example:

```
$js = elgg_get_simplecache_url('your_view.js');
$css = elgg_get_simplecache_url('your_view.css');
```

Page/elements/foot vs footer

`page/elements/footer` is the content that goes inside this part of the page:

```
<div class="elgg-page-footer">
        <div class="elgg-inner">
                <!-- page/elements/footer goes here -->
        </div>
</div>
```

It's content is visible to end users and usually where you would put a sitemap or other secondary global navigation, copyright info, powered by elgg, etc.

`page/elements/foot` is inserted just before the ending `</body>` tag and is mostly meant as a place to insert scripts that don't already work with `elgg_register_js(array('location' => 'footer'));` or `elgg_require_js('amd/module');`. In other words, you should never override this view and probably don't need to extend it either. Just use the `elgg_*_js` functions instead

3.26 Artilugios

Los artilugios son zonas de contenido que los usuarios pueden arrastrar por su página para personalizar su disposición. Normalmente su dueño puede personalizarlos para mostrar más o menos contenido, y controlar quien puede verlos. De manera predeterminada, Elgg ofrece complementos para personalizar la página de perfil y de inicio mediante artilugios.

Por hacer: captura de pantalla.

Contents

3.26.1 Estructura

Para crear un artilugio, cree dos vistas:

- `widgets/widget/edit`
- `widgets/widget/content`

`content.php` is responsible for all the content that will output within the widget. The `edit.php` file contains any extra edit functions you wish to present to the user. You do not need to add access level as this comes as part of the widget framework.

Nota: Using HTML checkboxes to set widget flags is problematic because if unchecked, the checkbox input is omitted from form submission. The effect is that you can only set and not clear flags. The «input/checkboxes» view will not work properly in a widget's edit panel.

3.26.2 Register the widget

Once you have created your edit and view pages, you need to initialize the plugin widget. This is done within the plugins `init()` function.

```
// Add generic new file widget
elgg_register_widget_type([
    'id' => 'filerepo',
    'name' => elgg_echo('widgets:filerepo:name'),
    'description' => elgg_echo('widgets:filerepo:description'),
]);
```

Nota: The only required attribute is the `id`.

Varios artilugios

It is possible to add multiple widgets for a plugin. You just initialize as many widget directories as you need.

```
 // Add generic new file widget
 elgg_register_widget_type([
     'id' => 'filerepo',
     'name' => elgg_echo('widgets:filerepo:name'),
     'description' => elgg_echo('widgets:filerepo:description'),
 ]);

 // Add a second file widget
elgg_register_widget_type([
     'id' => 'filerepo2',
     'name' => elgg_echo('widgets:filerepo2:name'),
     'description' => elgg_echo('widgets:filerepo2:description'),
 ]);

 // Add a third file widget
elgg_register_widget_type([
     'id' => 'filerepo3',
     'name' => elgg_echo('widgets:filerepo3:name'),
```

(continué en la próxima página)

(proviene de la página anterior)

```
    'description' => elgg_echo('widgets:filerepo3:description'),
]);
```

Make sure you have the corresponding directories within your plugin views structure:

```
'Plugin'
    /views
        /default
            /widgets
               /filerepo
                  /edit.php
                  /content.php
               /filerepo2
                  /edit.php
                  /content.php
               /filerepo3
                  /edit.php
                  /content.php
```

Magic widget name and description

When registering a widget you can omit providing a name and a description. If a translation in the following format is provided, they will be used. For the name: `widgets:<widget_id>:name` and for the description `widgets:<widget_id>:description`. If you make sure these translation are available in a translation file, you have very little work registering the widget.

```
elgg_register_widget_type(['id' => 'filerepo']);
```

How to restrict where widgets can be used

The widget can specify the context that it can be used in (all, just profile, just dashboard, etc.). If you do not specify a context they will be available for all contexts.

```
elgg_register_widget_type([
    'id' => 'filerepo',
    'context' => ['profile', 'dashboard', 'other_context'],
]);
```

Allow multiple widgets on the same page

By default you can only add one widget of the same type on the page. If you want more of the same widget on the page, you can specify this when registering the widget:

```
elgg_register_widget_type([
    'id' => 'filerepo',
    'multiple' => true,
]);
```

Register widgets in a hook

If, for example, you wish to conditionally register widgets you can also use a hook to register widgets.

```
function my_plugin_init() {
    elgg_register_plugin_hook_handler('handlers', 'widgets', 'my_plugin_conditional_
↪widgets_hook');
}

function my_plugin_conditional_widgets_hook($hook, $type, $return, $params) {
    if (!elgg_is_active_plugin('file')) {
        return;
    }

    $return[] = \Elgg\WidgetDefinition::factory([
        'id' => 'filerepo',
    ]);

    return $return;
}
```

Modify widget properties of existing widget registration

If, for example, you wish to change the allowed contexts of an already registered widget you can do so by re-registering the widget with `elgg_register_widget_type` as it will override an already existing widget definition. If you want even more control you can also use the `handlers, widgets` hook to change the widget definition.

```
function my_plugin_init() {
    elgg_register_plugin_hook_handler('handlers', 'widgets', 'my_plugin_change_widget_
↪definition_hook');
}

function my_plugin_change_widget_definition_hook($hook, $type, $return, $params) {
    foreach ($return as $key => $widget) {
        if ($widget->id === 'filerepo') {
            $return[$key]->multiple = false;
        }
    }

    return $return;
}
```

3.26.3 Default widgets

Si su complemento utiliza el canvas del artilugio, puede registrar que ofrece artilugios predeterminados en el núcleo de Elgg, para dejar que Elgg se ocupe de todo lo demás.

Para anunciar que su complemento ofrece artilugios predeterminados, registre un manejador para el gancho de complementos `get_list, default_widgets`:

```
elgg_register_plugin_hook_handler('get_list', 'default_widgets', 'my_plugin_default_
↪widgets_hook');
```

En el manejador, devuelva un vector que indique que ofrece artilugios predeterminados y cuándo crear los artilugios predeterminados. El vector puede contener las siguientes claves:

- `name` es el nombre de la página de artilugios. Ésta se muestra en la pestaña de la interfaz de administración.
- `widget_context` es el contexto desde el que se llama a la página de artilugios. Si no se indica de manera explícita, su valor será el identificador del complemento.

- `widget_columns` es el número de columnas que usará la página de artilugios.
- event - The Elgg event to create new widgets for. This is usually `create`.
- `entity_type` es el tipo de la entidad para la que crear los nuevos artilugios.
- `entity_subtype` es el subtipo de la entidad para la que crear los nuevos artilugios. Su valor puede ser `ELGG_ENTITIES_ANY_VALUE` para crearlo para todos los tipos de entidad.

Cuando un objeto desencadena un evento que coincide con el valor de los parámetros `event`, `entity_type` y `entity_subtype` que se pasen, el núcleo de Elgg buscará artilugios predeterminados que coincidan con el valor de `widget_context` indicado, y los copiará en las propiedades `owner_guid` y `container_guid` del objeto. También se copiará la configuración del artilugio.

```
function my_plugin_default_widgets_hook($hook, $type, $return, $params) {
    $return[] = array(
        'name' => elgg_echo('my_plugin'),
        'widget_context' => 'my_plugin',
        'widget_columns' => 3,

        'event' => 'create',
        'entity_type' => 'user',
        'entity_subtype' => ELGG_ENTITIES_ANY_VALUE,
    );

    return $return;
}
```

3.27 Walled Garden

Elgg supports a «Walled Garden» mode. In this mode, almost all pages are restricted to logged in users. This is useful for sites that don't allow public registration.

3.27.1 Activating Walled Garden mode

To activate Walled Garden mode in Elgg 1.8, go to the Administration section. On the right sidebar menu, under the «Configure» section, expand «Settings,» then click on «Advanced.»

From the Advanced Settings page, find the option labelled «Restrict pages to logged-in users.» Enable this option, then click «Save» to switch your site into Walled Garden mode.

3.27.2 Exposing pages through Walled Gardens

Many plugins extend Elgg by adding pages. Walled Garden mode will prevent these pages from being viewed by logged out users. Elgg uses *plugin hook* to manage which pages are visible through the Walled Garden.

Plugin authors must register pages as public if they should be viewable through Walled Gardens by responding to the `public_pages`, `walled_garden` plugin hook.

The returned value is an array of regexp expressions for public pages.

The following code shows how to expose http://example.org/my_plugin/public_page through a Walled Garden. This assumes the plugin has registered a *Page handler* for `my_plugin`.

```
elgg_register_plugin_hook_handler('public_pages', 'walled_garden', 'my_plugin_walled_
↪garden_public_pages');

function my_plugin_walled_garden_public_pages($hook, $type, $pages) {
   $pages[] = 'my_plugin/public_page';
   return $pages;
}
```

3.28 Servicios web

Construya una API HTTP para su sitio.

Elgg ofrece una infraestructura perfecta para construir servicios web. Esto permite a los desarrolladores exponer funcionalidad del sitio a otros sitios web y aplicaciones, así como a sitios web y aplicaciones de terceros. Aunque definimos la API como RESTful, en realidad es un híbrido entre REST y RPC, similar a las API de sitios como Flickr o Twitter.

To create an API for your Elgg site, you need to do 4 things:

- enable the web services plugin
- Exponer métodos.
- Configurar la autenticación de la API.
- Configurar la autenticación de usuarios.

Además, puede que desee controlar los tipos de autenticación disponibles en el sitio. Esto, también, se explica a continuación.

Contents

3.28.1 Seguridad

It is crucial that the web services are consumed via secure protocols. Do not enable web services if your site is not served via HTTPs. This is especially important if you allow API key only authentication.

If you are using third-party tools that expose API methods, make sure to carry out a thorough security audit. You may want to make sure that API authentication is required for ALL methods, even if they require user authentication. Methods that do not require API authentication can be easily abused to spam your site.

Ensure that the validity of API keys is limited and provide mechanisms for your API clients to renew their keys.

3.28.2 Exponer métodos

The function to use to expose a method is `elgg_ws_expose_function()`. As an example, let's assume you want to expose a function that echos text back to the calling application. The function could look like this

```
function my_echo($string) {
    return $string;
}
```

Dado que ofrecemos esta función para permitir a los desarrolladores probar sus clientes para la API, no necesitaremos ni autenticación de la API ni autenticación de usuarios. La siguiente llamada registra la función en la infraestructura de la API para servicios web:

```
elgg_ws_expose_function(
        "test.echo",
        "my_echo",
        [
                "string" => [
                        'type' => 'string',
                ]
        ],
        'A testing method which echos back a string',
        'GET',
        false,
        false
);
```

If you add this code to a plugin and then go to http://yoursite.com/services/api/rest/json/?method=system.api.list, you should now see your test.echo method listed as an API call. Further, to test the exposed method from a web browser, you could hit the url: http://yoursite.com/services/api/rest/json/?method=test.echo&string=testing and you should see JSON data like this:

```
{"status":0,"result":"testing"}
```

Plugins can filter the output of individual API methods by registering a handler for `'rest:output',$method` plugin hook.

Formatos de respuesta

JSON is the default format, however XML and serialized PHP can be fetched by enabling the `data_views` plugin and substituting `xml` or `php` in place of `json` in the above URLs.

You can also add additional response formats by defining new viewtypes.

Parameters

Parameters expected by each method should be listed as an associative array, where the key represents the parameter name, and the value contains an array with `type`, `default` and `required` fields.

Values submitted with the API request for each parameter should match the declared type. API will throw on exception if validation fails.

Recognized parameter types are:

- `integer` (or `int`)
- `boolean` (or `bool`)
- `string`
- `float`
- `array`

Unrecognized types will throw an API exception.

You can use additional fields to describe your parameter, e.g. `description`.

```
elgg_ws_expose_function(
        'test.greet',
        'my_greeting',
        [
                'name' => [
                        'type' => 'string',
                        'required' => true,
                        'description' => 'Name of the person to be greeted by the API
↪',
                ],
                'greeting' => [
                        'type' => 'string',
                        'required' => false,
                        'default' => 'Hello',
                        'description' => 'Greeting to be used, e.g. "Good day" or "Hi"
↪',
                ],
        ],
        'A testing method which greets the user with a custom greeting',
        'GET',
        false,
        false
);
```

Nota: If a missing parameter has no default value, the argument will be `null`. Before Elgg v2.1, a bug caused later arguments to be shifted left in this case.

Receive parameters as associative array

If you have a large number of method parameters, you can force the execution script to invoke the callback function with a single argument that contains an associative array of parameter => input pairs (instead of each parameter being a separate argument). To do that, set `$assoc` to `true` in `elgg_ws_expose_function()`.

```
function greet_me($values) {
        $name = elgg_extract('name', $values);
        $greeting = elgg_extract('greeting', $values, 'Hello');
        return "$greeting, $name";
}

elgg_ws_expose_function(
        "test.greet",
        "greet_me",
        [
                "name" => [
                        'type' => 'string',
                ],
                "greeting" => [
                        'type' => 'string',
                        'default' => 'Hello',
                        'required' => false,
                ],
        ],
        'A testing method which echos a greeting',
        'GET',
        false,
        false,
        true // $assoc makes the callback receive an associative array
);
```

Nota: If a missing parameter has no default value, `null` will be used.

3.28.3 Autenticación de la API

Puede que le interese controlar el acceso a algunas de la funciones que expone. Puede que esté exponiendo funciones para poder integrar Elgg con otra plataforma libre en el mismo servidor. En este caso, sólo quiere permitir que esa otra aplicación pueda acceder a estos métodos. Otra posibilidad es que limite qué desarrolladores externos tienen acceso a la API. O quizá quiere limitar el número de llamadas a la API que los desarrolladores pueden hacer cada día.

En todos estos casos, puede usar las funciones de autenticación de la API para controlar el acceso. Elgg provide dos métodos de serie para realizar autenticaciones contra la API: mediante una clave y mediante una firma HMAC. También puede incluir sus propios métodos de autenticación. El método de autenticación mediante una clave es muy similar al que usan servicios como Google, Flickr o Twitter. Los desarrolladores pueden solicitar una clave (una cadena de texto aleatoria) y incluir la clave en las llamadas a la API que requieren autenticación. Las claves se almacenan en una base de datos y si una llamada a la API no incluye la clave o incluye una clave incorrecta, la llamada a la API se contesta con un mensaje de error.

Autenticación mediante clave

Por ejemplo, escribamos una función que devuelva el número de usuarios que han visitado el sitio durante los últimos x minutos:

```
function count_active_users($minutes=10) {
    $seconds = 60 * $minutes;
    $count = count(find_active_users($seconds, 9999));
    return $count;
}
```

Ahora, expongamos la función y convirtamos el número de minutos en un parámetro opcional:

```
elgg_ws_expose_function(
        "users.active",
        "count_active_users",
        [
                "minutes" => [
                        'type' => 'int',
                        'required' => false,
                ],
        ],
        'Number of users who have used the site in the past x minutes',
        'GET',
        true,
        false
);
```

La función está ahora disponible, y si consulta `system.api.list` podrá comprobar que la función requiere autenticación contra la API. Si intenta acceder al método mediante un navegador web, obtendrá un mensaje de error que le informará de que no ha sido posible autenticar la llamada contra la API. Para probar el método, necesita una clave de la API. Por suerte, existe un complemento, «apiadmin», que creará una clave para usted. El complemento está disponible en el repositorio de complementos de Elgg. El complemento ofrece dos claves, una pública y otra privada, y usted podrá utilizar la pública para autenticarse contra la API por este método. Obtenga una clave y realice una solicitud de tipo GET a esta función con su navegador web pasándole a la función la clave como valor del parámetro `api_key`. Este podría ser su aspecto: http://yoursite.com/services/api/rest/xml/?method=users.active&api_key=1140321cb56c71710c38feefdf72bc462938f59f.

Autenticación mediante firma

The *HMAC Authentication* is similar to what is used with OAuth or Amazon's S3 service. This involves both the public and private key. If you want to be very sure that the API calls are coming from the developer you think they are coming from and you want to make sure the data is not being tampered with during transmission, you would use this authentication method. Be aware that it is much more involved and could turn off developers when there are other sites out there with key-based authentication.

3.28.4 Autenticación de usuarios

De momento hemos estado permitiendo a los desarrolladores obtener datos del sitio. Ahora procederemos a permitirles enviar datos. En este caso, los datos se añadirán de parte de un usuario. Imagine que ha creado una aplicación que permite a los usuarios publicar en el Wire y que usted necesita asegurarse de que un usuario no publica nada desde la cuenta de otra persona. Elgg ofrece un método de autenticación de usuarios mediante códigos aleatorios (tokens). Este método permite que un usuario envíe su nombre de usuario y contraseña mediante el método `auth.gettoken` y a cambio obtenga un código. Durante un cierto período de tiempo, el usuario puede usar el código recibido para autenticar todas las llamadas a la API antes de que el código caduque, pasando el código como valor del parámetro `auth_token`. Si no quiere que los usuarios les faciliten sus contraseñas a aplicaciones de terceros, también puede extender la funcionalidad actual para utilizar un método como OAuth.

Escribamos la función para publicar en el Wire:

```
function my_post_to_wire($text) {

    $text = substr($text, 0, 140);

    $access = ACCESS_PUBLIC;
```

(continué en la próxima página)

(proviene de la página anterior)

```
    // returns guid of wire post
    return thewire_save_post($text, $access, "api");
}
```

Expondremos la función como ya hemos hecho anteriormente, con la diferencia de que esta vez exigiremos autenticación de usuario, y que la solicitud sea de tipo POST en vez de GET.

```
elgg_ws_expose_function(
        "thewire.post",
        "my_post_to_wire",
        [
                "text" => [
                        'type' => 'string',
                ],
        ],
        'Post to the wire. 140 characters or less',
        'POST',
        true,
        true
);
```

Please note that you will not be able to test this using a web browser as you did with the other methods. You need to write some client code to do this.

3.28.5 Expandir la API

En cuanto se habitúe a la infraestructura del API para servicios web de Elgg, querrá proceder a diseñar su API: decidir qué datos quiere exponer, quién y en qué consistirán los usuarios de la API, cómo accederán a las claves de autenticación, cómo escribirá la documentación del API, etc. Asegúrese de echarle una ojeada a las API creadas por sitios web 2.0 populares, a modo de inspiración. Si pretende que desarrolladores de terceros construyan aplicaciones utilizando su API, quizá debería plantearse ofrecer uno o varios clientes específicos de algún lenguaje de programación.

3.28.6 Determinar el método de autenticación disponible

La API para servicios web de Elgg usa un tipo de arquitectura de módulo de autenticación conectable («PAM» por sus siglas en inglés) para gestionar la forma en que se autentifican los usuarios y desarrolladores. Esto le ofrece flexibilidad para añadir y eliminar módulos de autenticación. ¿No quiere usar el PAM predeterminado de autenticación de usuarios sino que prefiere usar OAuth? Pues puede hacerlo.

El primer paso consiste en registrar una llamada de retorno (callback) para el gancho de complementos *rest, init*:

```
register_plugin_hook('rest', 'init', 'rest_plugin_setup_pams');
```

A continuación, en la función de llamada de retorno registre los PAM que quiera utilizar:

```
function rest_plugin_setup_pams() {
    // user token can also be used for user authentication
    register_pam_handler('pam_auth_usertoken');

    // simple API key check
    register_pam_handler('api_auth_key', "sufficient", "api");
```

(continué en la próxima página)

(proviene de la página anterior)

```
    // override the default pams
    return true;
}
```

A la hora de hacer pruebas, puede que le resulte útil registrar el PAM `pam_auth_session` para poder probar fácilmente sus métodos desde el navegador web. Pero tenga cuidado de no usar este PAM en un sitio de producción, pues podría exponer a sus usuarios a un ataque de falsificación de peticiones entre sitios distintos.

3.28.7 Related

HMAC Authentication

Elgg's RESTful API framework provides functions to support a HMAC signature scheme for API authentication. The client must send the HMAC signature together with a set of special HTTP headers when making a call that requires API authentication. This ensures that the API call is being made from the stated client and that the data has not been tampered with.

The HMAC must be constructed over the following data:

- The public API key identifying you to the Elgg api server as provided by the APIAdmin plugin
- The private API Key provided by Elgg (that is companion to the public key)
- The current unix time in seconds
- A nonce to guarantee two requests the same second have different signatures
- URL encoded string representation of any GET variable parameters, eg `method=test.test&foo=bar`
- If you are sending post data, the hash of this data

Some extra information must be added to the HTTP header in order for this data to be correctly processed:

- **X-Elgg-apikey** - The public API key
- **X-Elgg-time** - Unix time used in the HMAC calculation
- **X-Elgg-none** - a random string
- **X-Elgg-hmac** - The HMAC as base64 encoded
- **X-Elgg-hmac-algo** - The algorithm used in the HMAC calculation - eg, sha1, md5 etc.

If you are sending POST data you must also send:

- **X-Elgg-posthash** - The hash of the POST data
- **X-Elgg-posthash-algo** - The algorithm used to produce the POST data hash - eg, md5
- **Content-type** - The content type of the data you are sending (if in doubt use application/octet-stream)
- **Content-Length** - The length in bytes of your POST data

Elgg provides a sample API client that implements this HMAC signature: send_api_call(). It serves as a good reference on how to implement it.

3.29 Actualizar complementos

Prepare su complemento para la siguiente versión de Elgg.

See the administrator guides for *how to upgrade a live site*.

Contents

- *Dropped `jquery-migrate` and upgraded `jquery` to ^2.1.4*
- *JS and CSS views have been moved out of the js/ and css/ directories*
- *`fxp/composer-asset-plugin` is now required to install Elgg from source*
- *List of deprecated views and view arguments that have been removed*
- *All scripts moved to bottom of page*
- *Attribute formatter removes keys with underscores*
- *Breadcrumbs*
- *Callbacks in Queries*
- *Comments plugin hook*
- *Container permissions hook*
- *Creating or deleting a relationship triggers only one event*
- *Discussion feature has been pulled from groups into its own plugin*
- *Dropped login-over-https feature*
- *Elgg has migrated from ext/mysql to PDO MySQL*
- *Event/Hook calling order may change*
- *`export/` URLs are no longer available*
- *Icons migrated to Font Awesome*
- *Increase of z-index value in elgg-menu-site class*
- *input/autocomplete view*
- *Introduced third-party library for sending email*
- *Label elements*
- *Plugin Aalborg Theme*
- *Plugin Likes*
- *Plugin Messages*
- *Plugin Blog*
- *Plugin Bookmarks*
- *Plugin File*
- *Removed Classes*
- *Removed keys available via `elgg_get_config()`*
- *Removed Functions*
- *Removed methods*
- *Removed Plugin Hooks*
- *Removed Actions*
- *Removed Views*
- *Removed View Variables*

3.29.1 From 2.2 to 2.3

PHP Version

PHP 5.5 has reached end of life in July 2016. To ensure that Elgg sites are secure, we now require PHP 5.6 for new installations.

Existing installations can continue using PHP 5.5 until Elgg 3.0.

In order to upgrade Elgg to 2.3 using composer while using PHP 5.5, you may need to use `--ignore-platform-reqs` flag.

Deprecated APIs

- Registering for `to:object` hook by the extender name: Use `to:object, annotation` and `to:object, metadata` hooks instead.
- `ajax_forward_hook()`: No longer used as handler for *"forward","all"* hook. Ajax response is now wrapped by the `ResponseFactory`
- `ajax_action_hook()`: No longer used as handler for *"action","all"* hook. Output buffering now starts before the hook is triggered in `ActionsService`
- `elgg_error_page_handler()`: No longer used as a handler for *"forward",<error_code>* hooks

- `get_uploaded_file()`: Use new file uploads API instead
- `get_user_notification_settings()`: Use `ElggUser::getNotificationSettings()`
- `set_user_notification_setting()`: Use `ElggUser::setNotificationSetting()`
- `pagesetup, system` event: Use the menu or page shell hooks instead.
- `elgg.walled_garden` JavaScript is deprecated: Use `elgg/walled_garden` AMD module instead.
- `elgg()->getDb()->getTableprefix()`: Use `elgg_get_config('dbprefix')`.
- Private `update_entity_last_action()`: Refrain from manually updating last action timestamp.
- Setting non-public `access_id` on metadata is deprecated. See below.
- `get_resized_image_from_existing_file()`: Use `elgg_save_resized_image()`.
- `get_resized_image_from_uploaded_file()`: Use `elgg_save_resized_image()` in combination with upload API.
- `get_image_resize_parameters()` will be removed.
- `elgg_view_input()`: Use `elgg_view_field()`. Apologies for the API churn.

Deprecated Views

- `resources/file/world`: Use the `resources/file/all` view instead.
- `resources/pages/world`: Use the `resources/pages/all` view instead.
- `walled_garden.js`: Use the `elgg/walled_garden` module instead.

New API for page and action handling

Page handlers and action script files should now return an instance of `\Elgg\Http\ResponseBuilder`. Plugins should use the following convenience functions to build responses:

- `elgg_ok_response()` sends a 2xx response with HTML (page handler) or JSON data (actions)
- `elgg_error_response()` sends a 4xx or 5xx response without content/data
- `elgg_redirect_response()` silently redirects the request

New API for working with file uploads

- `elgg_get_uploaded_files()` - returns an array of Symfony uploaded file objects
- `ElggFile::acceptUploadedFile()` - moves an uploaded file to Elgg's filestore

New API for manipulating images

New image manipulation service implements a more efficient approach to cropping and resizing images.

- `elgg_save_resized_image()` - crops and resizes an image to preferred dimensions

New API for events

- `elgg_clear_event_handlers()` - similar to `elgg_clear_plugin_hook_handlers` this functions removes all registered event handlers

New API for signing URLs

URLs can now be signed with a SHA-256 HMAC key and validated at any time before URL expiry. This feature can be used to tokenize action URLs in email notifications, as well as other uses outside of the Elgg installation.

- `elgg_http_get_signed_url()` - signs the URL with HMAC key
- `elgg_http_validate_signed_url()` - validates the signed URL
- `elgg_signed_request_gatekeeper()` - gatekeeper that validates the signature of the current request

Extendable form views

Form footer rendering can now be deferred until the form view and its extensions have finished rendering. This allows plugins to collaborate on form views without breaking the markup logic.

- `elgg_set_form_footer()` - sets form footer for deferred rendering
- `elgg_get_form_footer()` - returns currently set form footer

Metadata `access_id`

It's now deprecated to create metadata with an explicit `access_id` value other than `ACCESS_PUBLIC`.

In Elgg 3.0, metadata will not be access controlled, and will be available in all contexts. If your plugin relies on access control of metadata, it would be wise to migrate storage to annotations or entities instead.

New API for extracting class names from arrays

Similar to `elgg_extract()`, `elgg_extract_class()` extracts the «class» key (if present), merges into existing class names, and always returns an array.

Notificaciones

- A high level `'prepare','notification'` hook is now triggered for instant and subscription notifications and can be used to alter notification objects irrespective of their type.
- `'format','notification:<method>'` hook is now triggered for instant and subscription notifications and can be used to format the notification (e.g. strip HTML tags, wrap the notification body in a template etc).
- Instant notifications are now handled by the notifications service, hence almost all hooks applicable to subscription notifications also apply to instant notifications.
- `elgg_get_notification_methods()` can be used to obtain registered notification methods
- Added `ElggUser::getNotificationSettings()` and `ElggUser::setNotificationSetting()`

Entity list functions can output tables

In functions like `elgg_list_entities($options)`, table output is possible by setting `$options['list_type'] = 'table'` and providing an array of table columns as `$options['columns']`. Each column is an `Elgg\Views\TableColumn` object, usually created via methods on the service `elgg()->table_columns`.

Plugins can provide or alter these factory methods (see `Elgg\Views\TableColumn\ColumnFactory`). See the view `admin/users/newest` for a usage example.

Inline tabs components

Inline tabs component can now be rendered with `page/components/tabs` view. The components allows to switch between pre-poluated and ajax-loaded. See `page/components/tabs` in core views and `theme_sandbox/components/tabs` in developers plugin for usage instructions and examples.

API to alter registration and login URL

- `elgg_get_registration_url()` should be used to obtain site's registration URL
- `elgg_get_login_url()` should be used to obtain site's login URL
- `registration_url, site` hook can be used to alter the default registration URL
- `login_url, site` hook can be used to alter the default login URL

Support for fieldsets in forms

- `elgg_view_field()` replaces `elgg_view_input()`. It has a similar API, but accepts a single array.
- `elgg_view_field()` supports `#type`, `#label`, `#help` and `#class`, allowing unprefixed versions to be sent to the input view `$vars`.
- The new view `input/fieldset` can be used to render a set of fields, each rendered with `elgg_view_field()`.

3.29.2 From 2.1 to 2.2

Deprecated APIs

- `elgg.ui.river` JavaScript library: Remove calls to `elgg_load_js('elgg.ui.river')` from plugin code. Update `core/river/filter` and `forms/comment/save`, if overwritten, to require component AMD modules
- `elgg.ui.popupOpen()` and `elgg.ui.popupClose()` methods in `elgg.ui` JS library: Use `elgg/popup` module instead.
- `lightbox.js` library: Do not use `elgg_load_js('lightbox.js');` unless your code references deprecated `elgg.ui.lightbox` namespace. Use `elgg/lightbox` AMD module instead.
- `elgg.embed` library and `elgg.embed` object: Do not use `elgg_load_js('elgg.embed')`. Use `elgg/embed` AMD module instead
- Accessing `icons_sizes` config value directly: Use `elgg_get_icon_sizes()`
- `can_write_to_container()`: Use `ElggEntity::canWriteToContainer()`

Deprecated Views

- `elgg/ui.river.js` is deprecated: Do not rely on simplecache URLs to work.
- `groups/js` is deprecated: Use `groups/navigation` AMD module as a menu item dependency for «feature» and «unfeature» menu items instead.
- `lightbox/settings.js` is deprecated: Use `getOptions, ui.lightbox` JS plugin hook or `data-colorbox-opts` attribute.
- `elgg/ckeditor/insert.js` is deprecated: You no longer need to include it, hook registration takes place in `elgg/ckeditor` module
- `embed/embed.js` is deprecated: Use `elgg/embed` AMD module.

Added `elgg/popup` module

New *elgg/popup module* can be used to build out more complex trigger-popup interactions, including binding custom anchor types and opening/closing popups programmatically.

Added `elgg/lightbox` module

New *elgg/lightbox module* can be used to open and close the lightbox programmatically.

Added `elgg/embed` module

Even though rarely necessary, `elgg/embed` AMD module can be used to access the embed methods programmatically. The module bootstraps itself when necessary and is unlikely to require further decoration.

New API for handling entity icons

- `ElggEntity` now implements `\Elgg\EntityIcon` interface
- `elgg_get_icon_sizes()` - return entity type/subtype specific icon sizes
- `ElggEntity::saveIconFromUploadedFile()` - creates icons from an uploaded file
- `ElggEntity::saveIconFromLocalFile()` - creates icons from a local file
- `ElggEntity::saveIconFromElggFile()` - creates icons from an instance of `ElggFile`
- `ElggEntity::getIcon()` - returns an instanceof `ElggIcon` that points to entity icon location on filestore (this may be just a placeholder, use `ElggEntity::hasIcon()` to validate if file has been written)
- `ElggEntity::deleteIcon()` - deletes entity icons
- `ElggEntity::getIconLastChange()` - return modified time of the icon file
- `ElggEntity::hasIcon()` - checks if an icon with given size has been created
- `elgg_get_embed_url()` - can be used to return an embed URL for an entity's icon (served via */serve-icon* handler)

User avatars are now served via `serve-file` handler. Plugins should start using `elgg_get_inline_url()` and note that:

- `/avatar/view` page handler and resource view have been deprecated
- `/mod/profile/icondirect.php` file has been deprecated

- `profile_set_icon_url()` is no longer registered as a callback for `"entity:icon:url","user"` plugin hook

Group avatars are now served via `serve-file` handler. Plugins should start using `elgg_get_inline_url()` and note that:

- `groupicon` page handler (`groups_icon_handler()`) has been deprecated
- `/mod/groups/icon.php` file has been deprecated

File entity thumbs and downloads are now served via `serve-file` handler. Plugins should start using `elgg_get_inline_url()` and `elgg_get_download_url()` and note that:

- `file/download` page handler and resource view have been deprecated
- `mod/file/thumbnail.php` file has been deprecated
- Several views have been updated to use new download URLs, including:
  - `mod/file/views/default/file/specialcontent/audio/default.php`
  - `mod/file/views/default/file/specialcontent/image/default.php`
  - `mod/file/views/default/resources/file/view.php`
  - `mod/file/views/rss/file/enclosure.php`

Removed APIs

Just a warning that the private entity cache functions (e.g. `_elgg_retrieve_cached_entity`) have been removed. Some plugins may have been using them. Plugins should not use private APIs as they will more often be removed without notice.

Improved `elgg/ckeditor` module

elgg/ckeditor module can now be used to add WYSIWYG to a textarea programmatically with `elgg/ckeditor#bind`.

3.29.3 From 2.0 to 2.1

Deprecated APIs

- `ElggFile::setFilestore`
- `get_default_filestore`
- `set_default_filestore`
- `elgg_get_config('siteemail')`: Use `elgg_get_site_entity()->email`
- URLs starting with `/css/` and `/js/`: Use `elgg_get_simplecache_url()`
- `elgg.ui.widgets` JavaScript object is deprecated by `elgg/widgets` AMD module

`Application::getDb()` changes

If you're using this low-level API, do not expect it to return an `Elgg\Database` instance in 3.0. It now returns an `Elgg\Application\Database` with many deprecated. These methods were never meant to be made public API, but we will do our best to support them in 2.x.

Added `elgg/widgets` module

If your plugin code calls `elgg.ui.widgets.init()`, instead use the *elgg/widgets module*.

3.29.4 From 1.x to 2.0

Elgg can be now installed as a composer dependency instead of at document root

That means an Elgg site can look something like this:

```
settings.php
vendor/
  elgg/
    elgg/
      engine/
        start.php
      _graphics/
        elgg_sprites.png
mod/
  blog
  bookmarks
  ...
```

`elgg_get_root_path` and `$CONFIG->path` will return the path to the application root directory, which is not necessarily the same as Elgg core's root directory (which in this case is `vendor/elgg/elgg/`).

Do not attempt to access the core Elgg from your plugin directly, since you cannot rely on its location on the filesystem.

In particular, don't try load `engine/start.php`.

```
// Don't do this!
dirname(__DIR__) . "/engine/start.php";
```

To boot Elgg manually, you can use the class `Elgg\Application`.

```
// boot Elgg in mod/myplugin/foo.php
require_once dirname(dirname(__DIR__)) . '/vendor/autoload.php';
\Elgg\Application::start();
```

However, use this approach sparingly. Prefer *Encaminamiento* instead whenever possible as that keeps your public URLs and your filesystem layout decoupled.

Also, don't try to access the `_graphics` files directly.

```
readfile(elgg_get_root_path() . "_graphics/elgg_sprites.png");
```

Use *Vistas* instead:

```
echo elgg_view('elgg_sprites.png');
```

Cacheable views must have a file extension in their names

This requirement makes it possibile for us to serve assets directly from disk for performance, instead of serving them through PHP.

It also makes it much easier to explore the available cached resources by navigating to dataroot/views_simplecache and browsing around.

- Bad: `my/cool/template`
- Good: `my/cool/template.html`

We now cache assets by `"$viewtype/$view"`, not `md5("$viewtype|$view")`, which can result in conflicts between cacheable views that don't have file extensions to disambiguate files from directories.

Dropped `jquery-migrate` and upgraded `jquery` to ^2.1.4

jQuery 2.x is API-compatible with 1.x, but drops support for IE8-, which Elgg hasn't supported for some time anyways.

See http://jquery.com/upgrade-guide/1.9/ for how to move off jquery-migrate.

If you'd prefer to just add it back, you can use this code in your plugin's init:

```
elgg_register_js('jquery-migrate', elgg_get_simplecache_url('jquery-migrate.js'),
↪'head');
elgg_load_js('jquery-migrate');
```

Also, define a `jquery-migrate.js` view containing the contents of the script.

JS and CSS views have been moved out of the js/ and css/ directories

They also have been given .js and .css extensions respectively if they didn't already have them:

Old view	New view
`js/view`	`view.js`
`js/other.js`	`other.js`
`css/view`	`view.css`
`css/other.css`	`other.css`
`js/img.png`	`img.png`

The main benefit this brings is being able to co-locate related assets. So a template (`view.php`) can have its CSS/JS dependencies right next to it (`view.css`, `view.js`).

Care has been taken to make this change as backwards-compatible as possible, so you should not need to update any view references right away. However, you are certainly encouraged to move your JS and CSS views to their new, canonical locations.

Practically speaking, this carries a few gotchas:

The `view_vars, $view_name` and `view, $view_name` hooks will operate on the *canonical* view name:

```
elgg_register_plugin_hook_handler('view', 'css/elgg', function($hook, $view_name) {
  assert($view_name == 'elgg.css') // not "css/elgg"
});
```

Using the `view, all` hook and checking for individual views may not work as intended:

```
elgg_register_plugin_hook_handler('view', 'all', function($hook, $view_name) {
  // Won't work because "css/elgg" was aliased to "elgg.css"
  if ($view_name == 'css/elgg') {
    // Never executed...
  }

  // Won't work because no canonical views start with css/* anymore
```

(continué en la próxima página)

(proviene de la página anterior)

```
  if (strpos($view_name, 'css/') === 0) {
    // Never executed...
  }
});
```

Please let us know about any other BC issues this change causes. We'd like to fix as many as possible to make the transition smooth.

`fxp/composer-asset-plugin` is now required to install Elgg from source

We use `fxp/composer-asset-plugin` to manage our browser assets (js, css, html) with Composer, but it must be installed globally *before installing Elgg* in order for the `bower-asset/*` packages to be recognized. To install it, run:

```
composer global require fxp/composer-asset-plugin
```

If you don't do this before running `composer install` or `composer create-project`, you will get an error message:

```
[InvalidArgumentException]
Package fxp/composer-asset-plugin not found
```

List of deprecated views and view arguments that have been removed

We dropped support for and/or removed the following views:

- canvas/layouts/*
- categories
- categories/view
- core/settings/tools
- embed/addcontentjs
- footer/analytics (Use page/elements/foot instead)
- groups/left_column
- groups/right_column
- groups/search/finishblurb
- groups/search/startblurb
- input/calendar (Use input/date instead)
- input/datepicker (Use input/date instead)
- input/pulldown (Use input/select instead)
- invitefriends/formitems
- js/admin (Use AMD and `elgg_require_js` instead of extending JS views)
- js/initialise_elgg (Use AMD and `elgg_require_js` instead of extending JS views)
- members/nav
- metatags (Use the "head", "page" plugin hook instead)

- navigation/topbar_tools
- navigation/viewtype
- notifications/subscriptions/groupsform
- object/groupforumtopic
- output/calendar (Use output/date instead)
- output/confirmlink (Use output/url instead)
- page_elements/contentwrapper
- page/elements/shortcut_icon (Use the "head", "page" plugin hook instead)
- page/elements/wrapper
- profile/icon (Use `elgg_get_entity_icon`)
- river/object/groupforumtopic/create
- settings/{plugin}/edit (Use plugins/{plugin}/settings instead)
- user/search/finishblurb
- user/search/startblurb
- usersettings/{plugin}/edit (Use plugins/{plugin}/usersettings instead)
- widgets/{handler}/view (Use widgets/{handler}/content instead)

We also dropped the following arguments to views:

- «value» in output/iframe (Use «src» instead)
- «area2» and «area3» in page/elements/sidebar (Use «sidebar» or view extension instead)
- «js» in icon views (e.g. icon/user/default)
- «options» to input/radio and input/checkboxes which aren't key-value pairs will no longer be acceptable.

All scripts moved to bottom of page

You should test your plugin **with the JavaScript error console visible**. For performance reasons, Elgg no longer supports `script` elements in the `head` element or in HTML views. `elgg_register_js` will now load *all* scripts at the end of the `body` element.

You must convert inline scripts to *AMD* or to external scripts loaded with `elgg_load_js`.

Early in the page, Elgg provides a shim of the RequireJS `require()` function that simply queues code until the AMD `elgg` and `jQuery` modules are defined. This provides a straightforward way to convert many inline scripts to use `require()`.

Inline code which will fail because the stack is not yet loaded:

```
<script>
$(function () {
    // code using $ and elgg
});
</script>
```

This should work in Elgg 2.0:

```
<script>
require(['elgg', 'jquery'], function (elgg, $) {
    $(function () {
        // code using $ and elgg
    });
});
</script>
```

Attribute formatter removes keys with underscores

`elgg_format_attributes()` (and all APIs that use it) now filter out attributes whose name contains an underscore. If the attribute begins with `data-`, however, it will not be removed.

Breadcrumbs

Breadcrumb display now removes the last item if it does not contain a link. To restore the previous behavior, replace the plugin hook handler `elgg_prepare_breadcrumbs` with your own:

```
elgg_unregister_plugin_hook_handler('prepare', 'breadcrumbs', 'elgg_prepare_
↪breadcrumbs');
elgg_register_plugin_hook_handler('prepare', 'breadcrumbs', 'myplugin_prepare_
↪breadcrumbs');

function myplugin_prepare_breadcrumbs($hook, $type, $breadcrumbs, $params) {
    // just apply excerpt to titles
    foreach (array_keys($breadcrumbs) as $i) {
        $breadcrumbs[$i]['title'] = elgg_get_excerpt($breadcrumbs[$i]['title'], 100);
    }
    return $breadcrumbs;
}
```

Callbacks in Queries

Make sure to use only valid *callable* values for «callback» argument/options in the API.

Querying functions will now will throw a `RuntimeException` if `is_callable()` returns `false` for the given callback value. This includes functions such as `elgg_get_entities()`, `get_data()`, and many more.

Comments plugin hook

Plugins can now return an empty string from `'comments',$entity_type` hook in order to override the default comments component view. To force the default comments component, your plugin must return `false`. If you were using empty strings to force the default comments view, you need to update your hook handlers to return `false`.

Container permissions hook

The behavior of the `container_permissions_check` hook has changed when an entity is being created: Before 2.0, the hook would be called twice if the entity's container was not the owner. On the first call, the entity's owner would be passed in as `$params['container']`, which could confuse handlers.

In 2.0, when an entity is created in a container like a group, if the owner is the same as the logged in user (almost always the case), this first check is bypassed. So the `container_permissions_check` hook will almost always be called once with `$params['container']` being the correct container of the entity.

Creating or deleting a relationship triggers only one event

The «create» and «delete» relationship events are now only fired once, with `"relationship"` as the object type.

E.g. Listening for the `"create", "member"` or `"delete", "member"` event(s) will no longer capture group membership additions/removals. Use the `"create", "relationship"` or `"delete", "relationship"` events.

Discussion feature has been pulled from groups into its own plugin

The `object, groupforumtopic` subtype has been replaced with the `object, discussion` subtype. If your plugin is using or altering the old discussion feature, you should upgrade it to use the new subtype.

Nothing changes from the group owners" point of view. The discussion feature is still available as a group tool and all old discussions are intact.

Dropped login-over-https feature

For the best security and performance, serve all pages over HTTPS by switching the scheme in your site's wwwroot to `https` at http://yoursite.tld/admin/settings/advanced

Elgg has migrated from ext/mysql to PDO MySQL

Elgg now uses a `PDO_MYSQL` connection and no longer uses any ext/mysql functions. If you use `mysql_*` functions, implicitly relying on an open connection, these will fail.

If your code uses one of the following functions, read below.

- `execute_delayed_write_query()`
- `execute_delayed_read_query()`

If you provide a callable `$handler` to be called with the results, your handler will now receive a `\Doctrine\DBAL\Driver\Statement` object. Formerly this was an ext/mysql `result` resource.

Event/Hook calling order may change

When registering for events/hooks, the `all` keyword for wildcard matching no longer has any effect on the order that handlers are called. To ensure your handler is called last, you must give it the highest priority of all matching handlers, or to ensure your handler is called first, you must give it the lowest priority of all matching handlers.

If handlers were registered with the same priority, these are called in the order they were registered.

To emulate prior behavior, Elgg core handlers registered with the `all` keyword have been raised in priority. Some of these handlers will most likely be called in a different order.

`export/` URLs are no longer available

Elgg no longer provides this endpoint for exposing resource data.

Icons migrated to Font Awesome

Elgg's sprites and most of the CSS classes beginning with `elgg-icon-` have been removed.

Usage of `elgg_view_icon()` is backward compatible, but static HTML using the `elgg-icon` classes will have to be updated to the new markup.

Increase of z-index value in elgg-menu-site class

The value of z-index in the elgg-menu-site class has been increased from 1 to 50 to allow for page elements in the content area to use the z-index property without the «More» site menu's dropdown being displayed behind these elements. If your plugin/theme overrides the elgg-menu-site class or views/default/elements/navigation.css please adjust the z-index value in your modified CSS file accordingly.

input/autocomplete view

Plugins that override the `input/autocomplete` view will need to include the source URL in the `data-source` attribute of the input element, require the new `elgg/autocomplete` AMD module, and call its `init` method. The 1.x javascript library `elgg.autocomplete` is no longer used.

Introduced third-party library for sending email

We are using the excellent `Zend\Mail` library to send emails in Elgg 2.0. There are likely edge cases that the library handles differently than Elgg 1.x. Take care to test your email notifications carefully when upgrading to 2.0.

Label elements

The following views received `label` elements around some of the input fields. If your plugin/theme overrides these views please check for the new content.

- views/default/core/river/filter.php
- views/default/forms/admin/plugins/filter.php
- views/default/forms/admin/plugins/sort.php
- views/default/forms/login.php

Plugin Aalborg Theme

The view `page/elements/navbar` now uses a Font Awesome icon for the mobile menu selector instead of an image. The `bars.png` image and supporting CSS for the 1.12 rendering has been removed, so update your theme accordingly.

Plugin Likes

Objects are no longer likable by default. To support liking, you can register a handler to permit the annotation, or more simply register for the hook `["likes:is_likable", "<type>:<subtype>"]` and return true. E.g.

```
elgg_register_plugin_hook_handler('likes:is_likable', 'object:mysubtype',
↪'Elgg\Values::getTrue');
```

Just as before, the `permissions_check:annotate` hook is still called and may be used to override default behavior.

Plugin Messages

If you've removed or replaced the handler function `messages_notifier` to hide/alter the inbox icon, you'll instead need to do the same for the topbar menu handler `messages_register_topbar`. `messages_notifier` is no longer used to add the menu link.

Messages will no longer get the metadata "msg" for newly created messages. This means you can not rely on that metadata to exist.

Plugin Blog

The blog pages showing "Mine" or "Friends" listings of blogs have been changed to list all the blogs owned by the users (including those created in groups).

Plugin Bookmarks

The bookmark pages showing "Mine" or "Friends" listings of bookmarks have been changed to list all the bookmarks owned by the users (including those created in groups).

Plugin File

The file pages showing "Mine" or "Friends" listings of files have been changed to list all the files owned by the users (including those created in groups).

Removed Classes

- `ElggInspector`
- `Notable`
- `FilePluginFile`: replace with `ElggFile` (or load with `get_entity()`)

Removed keys available via `elgg_get_config()`

- `allowed_ajax_views`
- `dataroot_in_settings`
- `externals`
- `externals_map`
- `i18n_loaded_from_cache`
- `language_paths`
- `pagesetupdone`
- `registered_tag_metadata_names`
- `simplecache_enabled_in_settings`
- `translations`

- viewpath
- views
- view_path
- viewtype
- wordblacklist

Also note that plugins should not be accessing the global $CONFIG variable except for in settings.php.

Removed Functions

- blog_get_page_content_friends
- blog_get_page_content_read
- count_unread_messages()
- delete_entities()
- delete_object_entity()
- delete_user_entity()
- elgg_get_view_location()
- elgg_validate_action_url()
- execute_delayed_query()
- extend_view()
- get_db_error()
- get_db_link()
- get_entities()
- get_entities_from_access_id()
- get_entities_from_access_collection()
- get_entities_from_annotations()
- get_entities_from_metadata()
- get_entities_from_metadata_multi()
- get_entities_from_relationship()
- get_filetype_cloud()
- get_library_files()
- get_views()
- is_ip_in_array()
- list_entities()
- list_entities_from_annotations()
- list_group_search()
- list_registered_entities()
- list_user_search()

- `load_plugins()`
- `menu_item()`
- `make_register_object()`
- `mysql_*()`: Elgg *no longer uses ext/mysql*
- `remove_blacklist()`
- `search_for_group()`
- `search_for_object()`
- `search_for_site()`
- `search_for_user()`
- `search_list_objects_by_name()`
- `search_list_groups_by_name()`
- `search_list_users_by_name()`
- `set_template_handler()`
- `test_ip()`

Removed methods

- `ElggCache::set_variable()`
- `ElggCache::get_variable()`
- `ElggData::initialise_attributes()`
- `ElggData::getObjectOwnerGUID()`
- `ElggDiskFilestore::make_directory_root()`
- `ElggDiskFilestore::make_file_matrix()`
- `ElggDiskFilestore::user_file_matrix()`
- `ElggDiskFilestore::mb_str_split()`
- `ElggEntity::clearMetadata()`
- `ElggEntity::clearRelationships()`
- `ElggEntity::clearAnnotations()`
- `ElggEntity::getOwner()`
- `ElggEntity::setContainer()`
- `ElggEntity::getContainer()`
- `ElggEntity::getIcon()`
- `ElggEntity::setIcon()`
- `ElggExtender::getOwner()`
- `ElggFileCache::create_file()`
- `ElggObject::addToSite()`: parent function in ElggEntity still available
- `ElggObject::getSites()`: parent function in ElggEntity still available

- `ElggSite::getCollections()`
- `ElggUser::addToSite()`: parent function in ElggEntity still available
- `ElggUser::getCollections()`
- `ElggUser::getOwner()`
- `ElggUser::getSites()`: parent function in ElggEntity still available
- `ElggUser::listFriends()`
- `ElggUser::listGroups()`
- `ElggUser::removeFromSite()`: parent function in ElggEntity still available

The following arguments have also been dropped:

- `ElggSite::getMembers()` - 2: `$offset`
- `elgg_view_entity_list()` - 3: `$offset` - 4: `$limit` - 5: `$full_view` - 6: `$list_type_toggle` - 7: `$pagination`

Removed Plugin Hooks

- `[display, view]`: See the *new plugin hook*.

Removed Actions

- `widgets/upgrade`

Removed Views

- `forms/admin/plugins/change_state`

Removed View Variables

During rendering, the view system no longer injects these into the scope:

- `$vars['url']`: replace with `elgg_get_site_url()`
- `$vars['user']`: replace with `elgg_get_logged_in_user_entity()`
- `$vars['config']`: use `elgg_get_config()` and `elgg_set_config()`
- `$CONFIG`: use `elgg_get_config()` and `elgg_set_config()`

Also several workarounds for very old views are no longer performed. Make these changes:

- Set `$vars['full_view']` instead of `$vars['full']`.
- Set `$vars['name']` instead of `$vars['internalname']`.
- Set `$vars['id']` instead of `$vars['internalid']`.

Removed libraries

- `elgg:markdown`: Elgg no longer provides a markdown implementation. You must provide your own.

Specifying View via Properties

The metadata `$entity->view` no longer specifies the view used to render in `elgg_view_entity()`.

Similarly the property `$annotation->view` no longer has an effect within `elgg_view_annotation()`.

Viewtype is static after the initial `elgg_get_viewtype()` call

`elgg_set_viewtype()` must be used to set the viewtype at runtime. Although Elgg still checks the `view` input and `$CONFIG->view` initially, this is only done once per request.

Deprecations

It's deprecated to read or write to metadata keys starting with `filestore::` on `ElggFile` objects. In Elgg 3.0 this metadata will be deleted if it points to the current data root path, so few file objects will have it. Plugins should only use `ElggFile::setFilestore` if files need to be stored in a custom location.

Nota: This is not the only deprecation in Elgg 2.0. Plugin developers should watch their site error logs.

3.29.5 From 1.10 to 1.11

Comment highlighting

If your theme is using the file `views/default/css/elements/components.php`, you must add the following style definitions in it to enable highlighting for comments and discussion replies:

```
.elgg-comments .elgg-state-highlight {
        -webkit-animation: comment-highlight 5s;
        animation: comment-highlight 5s;
}
@-webkit-keyframes comment-highlight {
        from {background: #dff2ff;}
        to {background: white;}
}
@keyframes comment-highlight {
        from {background: #dff2ff;}
        to {background: white;}
}
```

3.29.6 From 1.9 to 1.10

File uploads

If your plugin is using a snippet copied from the `file/upload` action to fix detected mime types for Microsoft zipped formats, it can now be safely removed.

If your upload action performs other manipulations on detected mime and simple types, it is recommended to make use of available plugin hooks:

- `'mime_type','file'` for filtering detected mime types
- `'simple_type','file'` for filtering parsed simple types

3.29.7 De la versión 1.8 a la 1.9

En los ejemplos estamos actualizando un complemento imaginario, «Photos» (fotos).

Sólo se incluyen los cambios fundamentales. Por ejemplo, algunas de las funciones que están ahora obsoletas no se mencionan en esta sección.

Cada sección incluirá información sobre si el cambio es compatible hacia atrás con la versión 1.8 de Elgg.

El fichero «manifest»

Si su complemento es compatible con la versión 1.8 de Elgg, no necesita hacer ningún cambio en este fichero.

De todas formas, se recomienda añadir la etiqueta `<id>`. Su valor debería ser el nombre de la carpeta que contiene el complemento dentro de la carpeta `mod/`.

Si hace cambios que hacen el complemento incompatible con versiones anteriores con las que antes sí era compatible, debe actualizar la versión del complemento y la versión requerida de Elgg.

Ejemplo (resumido) de la versión anterior:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
    <name>Photos</name>
    <author>John Doe</author>
    <version>1.0</version>
    <description>Adds possibility to upload photos and arrange them into albums.</
↪description>
    <requires>
        <type>elgg_release</type>
        <version>1.8</version>
    </requires>
</plugin_manifest>
```

Ejemplo (resumido) de la nueva versión:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
    <name>Photos</name>
    <id>photos</id>
    <author>John Doe</author>
    <version>2.0</version>
    <description>Adds possibility to upload photos and arrange them into albums.</
↪description>
    <requires>
        <type>elgg_release</type>
        <version>1.9</version>
    </requires>
</plugin_manifest>
```

$CONFIG y $vars[“config”]

Tanto la variable global `$CONFIG` como el parámetro `$vars['config']` están obsoletos. Deberían substituirse por la función `elgg_get_config()`.

Ejemplo de código de la versión anterior:

```
// Using the global $CONFIG variable:
global $CONFIG;
$plugins_path = $CONFIG->plugins_path

// Using the $vars view parameter:
$plugins_path = $vars['plugins_path'];
```

Ejemplo de código de la nueva versión:

```
$plugins_path = elgg_get_config('plugins_path');
```

Nota: Compatible con la versión 1.8.

Nota: Así se actualizó el complemento «community_plugins»: https://github.com/Elgg/community_plugins/commit/f233999bbd1478a200ee783679c2e2897c9a0483

Ficheros de idioma

En la versión 1.8 de Elgg, los ficheros de idioma necesitaban usar la función `add_translation()`. A partir de la versión 1.9, basta con devolver el vector que previamente se pasó a la función como parámetro. El núcleo de Elgg se basará en el nombre del fichero (por ejemplo: «es.php») para determinar qué idioma contiene el fichero.

Ejemplo del método anterior en `languages/en.php`:

```
$english = array(
    'photos:all' => 'All photos',
);
add_translation('en', $english);
```

Ejemplo del nuevo método:

```
return array(
    'photos:all' => 'All photos',
);
```

Advertencia: No es compatible con la versión 1.8.

Notificaciones

Uno de los cambios más importantes en la versión 1.9 de Elgg es el del sistema de notificaciones. El nuevo sistema de notificaciones ofrece formas más flexibles y escalables de enviar notificaciones.

Ejemplo del método anterior:

```
function photos_init() {
    // Tell core that we want to send notifications about new photos
    register_notification_object('object', 'photo', elgg_echo('photo:new'));

    // Register a handler that creates the notification message
```

(continué en la próxima página)

(proviene de la página anterior)

```
    elgg_register_plugin_hook_handler('notify:entity:message', 'object', 'photos_
↪notify_message');
}

/**
 * Set the notification message body
 *
 * @param string $hook    Hook name
 * @param string $type    Hook type
 * @param string $message The current message body
 * @param array  $params  Parameters about the photo
 * @return string
 */
function photos_notify_message($hook, $type, $message, $params) {
    $entity = $params['entity'];
    $to_entity = $params['to_entity'];
    $method = $params['method'];
    if (elgg_instanceof($entity, 'object', 'photo')) {
        $descr = $entity->excerpt;
        $title = $entity->title;
        $owner = $entity->getOwnerEntity();
        return elgg_echo('photos:notification', array(
            $owner->name,
            $title,
            $descr,
            $entity->getURL()
        ));
    }
    return null;
}
```

Ejemplo del nuevo método:

```
function photos_init() {
    elgg_register_notification_event('object', 'photo', array('create'));
    elgg_register_plugin_hook_handler('prepare', 'notification:publish:object:photo',
↪'photos_prepare_notification');
}

/**
 * Prepare a notification message about a new photo
 *
 * @param string                           $hook         Hook name
 * @param string                           $type         Hook type
 * @param Elgg_Notifications_Notification $notification The notification to prepare
 * @param array                            $params       Hook parameters
 * @return Elgg_Notifications_Notification
 */
function photos_prepare_notification($hook, $type, $notification, $params) {
    $entity = $params['event']->getObject();
    $owner = $params['event']->getActor();
    $recipient = $params['recipient'];
    $language = $params['language'];
    $method = $params['method'];

    // Title for the notification
    $notification->subject = elgg_echo('photos:notify:subject', array($entity->title),
↪ $language);
```

(continué en la próxima página)

(proviene de la página anterior)

```
    // Message body for the notification
    $notification->body = elgg_echo('photos:notify:body', array(
        $owner->name,
        $entity->title,
        $entity->getExcerpt(),
        $entity->getURL()
    ), $language);

    // The summary text is used e.g. by the site_notifications plugin
    $notification->summary = elgg_echo('photos:notify:summary', array($entity->title),
↪ $language);

    return $notification;
}
```

Advertencia: No es compatible con la versión 1.8.

Nota: Así se actualizó el complemento «community_plugins» para usar el nuevo sistema: https://github.com/Elgg/community_plugins/commit/bfa356cfe8fb99ebbca4109a1b8a1383b70ff123

Notifications can also be sent with the `notify_user()` function.

It has however been updated to support three new optional parameters passed inside an array as the fifth parameter.

The parameters give notification plugins more control over the notifications, so they should be included whenever possible. For example the bundled site_notifications plugin won't work properly if the parameters are missing.

Parameters:

- **object** The object that we are notifying about (e.g. ElggEntity or ElggAnnotation). This is needed so that notification plugins can provide a link to the object.
- **action** String that describes the action that triggered the notification (e.g. «create», «update», etc).
- **summary** String that contains a summary of the notification. (It should be more informative than the notification subject but less informative than the notification body.)

Ejemplo del método anterior:

```
// Notify $owner that $user has added a $rating to an $entity created by him

$subject = elgg_echo('rating:notify:subject');
$body = elgg_echo('rating:notify:body', array(
        $owner->name,
        $user->name,
        $entity->title,
        $entity->getURL(),
));

notify_user($owner->guid,
                        $user->guid,
                        $subject,
                        $body
                );
```

Ejemplo del nuevo método:

```
// Notify $owner that $user has added a $rating to an $entity created by him

$subject = elgg_echo('rating:notify:subject');
$summary = elgg_echo('rating:notify:summary', array($entity->title));
$body = elgg_echo('rating:notify:body', array(
        $owner->name,
        $user->name,
        $entity->title,
        $entity->getURL(),
));

$params = array(
        'object' => $rating,
        'action' => 'create',
        'summary' => $summary,
);

notify_user($owner->guid,
                        $user->guid,
                        $subject,
                        $body,
                        $params
                );
```

Nota: Compatible con la versión 1.8.

Añadir elementos a la lista de actividad

```
add_to_river('river/object/photo/create', 'create', $user_guid, $photo_guid);
```

```
elgg_create_river_item(array(
    'view' => 'river/object/photo/create',
    'action_type' => 'create',
    'subject_guid' => $user_guid,
    'object_guid' => $photo_guid,
));
```

También puede hacer uso del parámetro opcional `target_guid` para indicar el destinatario de la acción de crear.

Por ejemplo, si la foto se pretendiese añadir a un álbum de fotos, se haría pasándole también:

```
'target_guid' => $album_guid,
```

Advertencia: No es compatible con la versión 1.8.

Gestores de direcciones URL de entidades

La función `elgg_register_entity_url_handler()` ha pasado a estar obsoleta en la versión 1.9 de Elgg, donde se recomienda usar el gancho de complementos `'entity:url', 'object'` en us lugar.

Ejemplo del método anterior:

```
/**
 * Initialize the photo plugin
 */
my_plugin_init() {
    elgg_register_entity_url_handler('object', 'photo', 'photo_url_handler');
}

/**
 * Returns the URL from a photo entity
 *
 * @param ElggEntity $entity
 * @return string
 */
function photo_url_handler($entity) {
    return "photo/view/{$entity->guid}";
}
```

Ejemplo del nuevo método:

```
/**
 * Initialize the photo plugin
 */
my_plugin_init() {
    elgg_register_plugin_hook_handler('entity:url', 'object', 'photo_url_handler');
}

/**
 * Returns the URL from a photo entity
 *
 * @param string $hook   'entity:url'
 * @param string $type   'object'
 * @param string $url    The current URL
 * @param array  $params Hook parameters
 * @return string
 */
function photo_url_handler($hook, $type, $url, $params) {
    $entity = $params['entity'];

    // Check that the entity is a photo object
    if ($entity->getSubtype() !== 'photo') {
        // This is not a photo object, so there's no need to go further
        return;
    }

    return "photo/view/{$entity->guid}";
}
```

Advertencia: No es compatible con la versión 1.8.

Servicios web

In Elgg 1.8 the web services API was included in core and methods were exposed using `expose_function()`. To enable the same functionality for Elgg 1.9, enable the «Web services 1.9» plugin and replace all calls to

`expose_function()` with `elgg_ws_expose_function()`.

3.29.8 De la versión 1.7 a la 1.8

La versión 1.8 ha sido el mayor salto en el desarrollo de Elgg desde la versión 1.0. Es por ello que para actualizar el núcleo de Elgg y sus complementos hace falta más trabajo que en anteriores actualizaciones. Se produjeron algunos cambios pequeños en la API, y siguiendo con nuestra práctica habitual, los métodos que pasaron a quedar obsoletos se actualizaron para funcionar con la nueva API. Los cambios más importantes afectaron a la estandarización de complementos y al sistema de vistas.

Actualizar el núcleo

Elimine las siguientes carpetas (se encuentran en el mismo nivel que «_graphics» o «engine»):

- _css
- account
- admin
- dashboard
- entities
- friends
- search
- settings
- simplecache
- views

Advertencia: Tendrá problemas si no elimina estas carpetas antes de actualizar.

Actualizar complementos

Use encaminamientos estándar con gestores de páginas

- Todo: /page_handler/all
- Contenido del usuario: /page_handler/owner/:username
- Contenido de contactos del usuario: /page_handler/friends/:username
- Entidad suelta: /page_handler/view/:guid/:title
- Añadido: /page_handler/add/:container_guid
- Edición: /page_handler/edit/:guid
- Lista de grupos: /page_handler/group/:guid/all

Incluya scripts de gestión de páginas desde el gestor de páginas

Casi todos los gestores de páginas deberían tener un script de gestión de páginas. Por ejemplo, `bookmarks/all => mod/bookmarks/pages/bookmarks/all.php`.

- Llame a `set_input()` para identificadores de entidades en el gestor de páginas y use `get_input()` en los scripts de gestión de páginas.
- Llame a `gatekeeper()` y a `admin_gatekeeper()` en la función del gestor de páginas si fuese necesario.
- La dirección URL del grupo debería usar el script `pages/:handler/owner.php`.
- Los gestores de páginas no deberían contener HTML.
- Actualice las direcciones URL en todo el complemento. No se olvide de eliminar `/pg/`.

Use gestores de páginas y scripts estandarizados

- Guarde los scripts de gestión de páginas en `mod/:plugin/pages/:page_handler/:page_name.php`
- Use la disposición de página del contenido en los scripts de gestión de páginas:

```
$content = elgg_view_layout('content', $options);
```

- Los scripts de gestión de páginas no deberían contener HTML.
- Llame a `elgg_push_breadcrumb()` en los scripts de gestión de páginas.
- No es necesario definir el dueño de una página si las direcciones URL están en el formato estándar.
- Para el contenido de grupos, compruebe el identificador del contenedor (`container_guid`) mediante la función `elgg_get_page_owner_entity()`.

La vista `object/:subtype`

- Asegúrese de que hay vistas para `$vars['full_view'] == true` y `$vars['full_view'] == false`. `$vars['full_view']` ha substituido a `$vars['full]`.
- Compruebe el objeto en `$vars['entity']`. Use `elgg_instance_of()` para asegurarse de que se trata de el tipo de entidad que busca.
- Devuelva `true` para cancelar la vista directamente si falta la entidad o ésta no es correcta.
- Use `elgg_view('object/elements/summary', array('entity' => $entity));` y `elgg_view_menu('entity', array('entity' => $entity));` como ayuda para dar formato. Debería usar poco o ningún lenguaje de etiquetas en estas vistas.

Actualizar la estructura de las acciones

- Use espacios de nombre para ficheros y nombres de acciones. Por ejemplo, `mod/blog/actions/blog/save.php` → `action/blog/save`.
- Use las siguientes direcciones URL de acciones:
  - Añadir: `action/:plugin/save`.
  - Editar: `action/:plugin/save`.

    - Eliminar: `action/:plugin/delete`.
- Haga que la acción de eliminar acepte `action/:handler/delete?guid=:guid` de forma que el menú de entidades de metadatos tenga la dirección URL correcta de manera predeterminada.

Actualice las funciones que se hayan quedado obsoletas

- Las funciones marcadas como obsoletas en la versión 1.7 de Elgg producen errores visibles en la versión 1.8.
- También puede actualizar funciones que se marcaron como obsoletas en la versión 1.8.
    - A muchas funciones de registro simplemente se les añadió el prefijo `elgg_` por motivos de consistencia, y actualizarlas no debería ser difícil.
    - Véase la lista completa en `/engine/lib/deprecated-1.8.php`.
    - Puede cambiar el nivel de depuración a «warning» (aviso) para recibir de manera visual recordatorios sobre el uso de funciones obsoletas.

Actualice las vistas de artilugios

Puede tomar como ejemplos los artilugios de blogs o de ficheros.

Actualice el módulo de perfiles de grupos

Use los complementos de blogs y de ficheros a modo de ejemplo. Le ayudarán a hacer posible cambiar el tema del complemento usando la nueva infraestructura CSS.

Actualice los formularios

- Mueva los cuerpos de los formularios a la vista `forms/:action` para usar el nuevo `elgg_view_form` de Evan.
- Use las vistas de introducción de datos en los cuerpos de los formularios en vez de usar HTML. Esto ayudará a que el complemento sea compatible con los temas, y a que sea más fácil de mantener ante cambios futuros.
- Añada una función que prepare el formulario. A modo de ejemplo, véase `mod/file/lib/file.php`.
- Persista los formularios. A modo de ejemplo, véanse la acción de enviar (upload) del complemento de ficheros y su función para preparar el formulario.

La API de los formularios se describe con más detalle en *Formularios y acciones*.

Haz limpieza de CSS y HTML

Se han añadido muchos patrones de CSS al fichero CSS base (módulos, bloques de imágenes, primitivas de espaciado). Se recomienda usar dichos patrones y clases siempre que sea posible. De esa forma:

1. Se reducen los costes de mantenimiento, puesto que puede eliminar la mayor parte de su código CSS personalizado.
2. Su complemento es más compatible con los temas de la comunidad.

Si necesita mucho código CSS, busque patrones que podrían añadirse al núcleo de Elgg.

Usamos guiones en vez de guiones bajos en las clases e identificadores, y le recomendamos que haga lo mismo por consistencia.

Si de verdad necesita su propio CSS, debería usar si propio espacio de nombres, uno distinto de `elgg-`.

Actualice el fichero manifest.xml

- Use http://el.gg/manifest17to18 para automatizar la tarea.
- No utilice la categoría «bundled» (de serie) en sus complementos. Esa categoría corresponde sólo a complementos distribuidos junto con Elgg.

Actualice las vistas de configuración global y de usuario

- La vista de configuración ha pasado a ser `plugins/:plugin/settings` (antes era `settings/:plugin/edit`).
- La vista de configuración de usuario ha pasado a ser `plugins/:plugin/usersettings` (antes era `usersettings/:plugin/edit`).

3.30 Lista de eventos fundamentales

Contents

3.30.1 Eventos del sistema

boot, system First event triggered. Triggered before plugins have been loaded.

plugins_boot, system Triggered just after the plugins are loaded. Rarely used. init, system is used instead.

init, system Plugins tend to use this event for initialization (extending views, registering callbacks, etc.)

ready, system Triggered after the `init, system` event. All plugins are fully loaded and the engine is ready to serve pages.

pagesetup, system (deprecated in 2.3) Called just before the first content is produced. Is triggered by `elgg_view()`. Use the menu or page shell hooks instead.

shutdown, system Triggered after the page has been sent to the user. Expensive operations could be done here and not make the user wait.

Nota: Depending upon your server configuration the PHP output might not be shown until after the process is completed. This means that any long-running processes will still delay the page load.

Nota: This event is prefered above using `register_shutdown_function` as you may not have access to all the Elgg services (eg. database) in the shutdown function but you will in the event.

---

regenerate_site_secret:before, system Return false to cancel regenerating the site secret. You should also provide a message to the user.

regenerate_site_secret:after, system Triggered after the site secret has been regenerated.

log, systemlog Called for all triggered events. Used internally by `system_log_default_logger()` to populate the `system_log` table.

upgrade, system Triggered after a system upgrade has finished. All upgrade scripts have run, but the caches are not cleared.

upgrade, upgrade

A single upgrade script finished executing. Handlers are passed a `stdClass` object with the properties

- from - The version of Elgg upgrading from.
- to - The version just upgraded to.

activate, plugin Return false to prevent activation of the plugin.

deactivate, plugin Return false to prevent deactivation of the plugin.

init:cookie, <name> Return false to override setting a cookie.

cache:flush, system Reset internal and external caches, by default including system_cache, simplecache, and memcache. One might use it to reset others such as APC, OPCache, or WinCache.

send:before, http_response Triggered before an HTTP response is sent. Handlers will receive an instance of *SymfonyComponentHttpFoundationResponse* that is to be sent to the requester. Handlers can terminate the event and prevent the response from being sent by returning *false*.

send:after, http_response Triggered after an HTTP response is sent. Handlers will receive an instance of *SymfonyComponentHttpFoundationResponse* that was sent to the requester.

3.30.2 Eventos de usuario

login:before, user Triggered during login. Returning false prevents the user from logging

login:after, user Triggered after the user logs in.

logout:before, user Triggered during logout. Returning false should prevent the user from logging out.

logout:after, user Triggered after the user logouts.

validate, user When a user registers, the user's account is disabled. This event is triggered to allow a plugin to determine how the user should be validated (for example, through an email with a validation link).

profileupdate, user User has changed profile

profileiconupdate, user User has changed profile icon

ban, user Triggered before a user is banned. Return false to prevent.

unban, user Triggered before a user is unbanned. Return false to prevent.

make_admin, user Triggered before a user is promoted to an admin. Return false to prevent.

remove_admin, user Triggered before a user is demoted from an admin. Return false to prevent.

3.30.3 Eventos de relaciones

create, relationship Triggered after a relationship has been created. Returning false deletes the relationship that was just created.

Nota: This event was broken in Elgg 1.9 - 1.12.3, returning false would *not* delete the relationship. This is working as of 1.12.4

---

delete, relationship Triggered before a relationship is deleted. Return false to prevent it from being deleted.

join, group Triggered after the user `$params['user']` has joined the group `$params['group']`.

leave, group Triggered before the user `$params['user']` has left the group `$params['group']`.

3.30.4 Eventos de entidades

create, <tipo de entidad> Triggered for user, group, object, and site entities after creation. Return false to delete entity.

update, <tipo de entidad> Triggered before an update for the user, group, object, and site entities. Return false to prevent update. The entity method `getOriginalAttributes()` can be used to identify which attributes have changed since the entity was last saved.

update:after, <entity type> Triggered after an update for the user, group, object, and site entities. The entity method `getOriginalAttributes()` can be used to identify which attributes have changed since the entity was last saved.

delete, <tipo de entidad> Triggered before entity deletion. Return false to prevent deletion.

disable, <tipo de entidad> Triggered before the entity is disabled. Return false to prevent disabling.

disable:after, <entity type> Triggered after the entity is disabled.

enable, <tipo de entidad> Return false to prevent enabling.

enable:after, <entity type> Triggered after the entity is enabled.

3.30.5 Eventos de metadatos

create, metadata Called after the metadata has been created. Return false to delete the metadata that was just created.

update, metadata Called after the metadata has been updated. Return false to *delete the metadata.*

delete, metadata Called before metadata is deleted. Return false to prevent deletion.

enable, metadata Called when enabling metadata. Return false to prevent enabling.

disable, metadata Called when disabling metadata. Return false to prevent disabling.

3.30.6 Eventos de anotaciones

annotate, <tipo de entidad> Called before the annotation has been created. Return false to prevent annotation of this entity.

create, annotation Called after the annotation has been created. Return false to delete the annotation.

update, annotation Called after the annotation has been updated. Return false to *delete the annotation.*

delete, annotation Called before annotation is deleted. Return false to prevent deletion.

enable, annotation Called when enabling annotations. Return false to prevent enabling.

disable, annotations Called when disabling annotations. Return false to prevent disabling.

3.30.7 River events

created, river Called after a river item is created.

> **Nota:** Use the plugin hook `creating, river` to cancel creation (or alter options).

delete:before, river Triggered before a river item is deleted. Returning false cancels the deletion.

delete:after, river Triggered after a river item was deleted.

3.30.8 File events

upload:after, file Called after an uploaded file has been written to filestore. Receives an instance of `ElggFile` the uploaded file was written to. The `ElggFile` may or may not be an entity with a GUID.

3.30.9 Notas

Because of bugs in the Elgg core, some events may be thrown more than once on the same action. For example, `update, object` is thrown twice.

3.31 Lista de ganchos de complementos en el núcleo

Contents

3.31.1 Ganchos del sistema

page_owner, system Filter the page_owner for the current page. No options are passed.

siteid, system

gc, system Allows plugins to run garbage collection for `$params['period']`.

unit_test, system Add a Simple Test test. (Deprecated.)

diagnostics:report, system Filter the output for the diagnostics report download.

search_types, get_types

cron, <periodo> Triggered by cron for each period.

validate, input Filter GET and POST input. This is used by `get_input()` to sanitize user input.

geocode, location Deprecated as of 1.9.

diagnostics:report, system Filters the output for a diagnostic report.

debug, log Triggered by the Logger. Return false to stop the default logging method. `$params` includes:

- **level - The debug level. One of:**
  - `Elgg_Logger::OFF`
  - `Elgg_Logger::ERROR`
  - `Elgg_Logger::WARNING`
  - `Elgg_Logger::NOTICE`
  - `Elgg_Logger::INFO`
- msg - The message
- display - Should this message be displayed?

format, friendly:title Formats the «friendly» title for strings. This is used for generating URLs.

format, friendly:time Formats the «friendly» time for the timestamp `$params['time']`.

format, strip_tags Filters a string to remove tags. The original string is passed as `$params['original_string']` and an optional set of allowed tags is passed as `$params['allowed_tags']`.

output:before, page In `elgg_view_page()`, this filters `$vars` before it's passed to the page shell view (`page/<page_shell>`). To stop sending the X-Frame-Options header, unregister the handler `_elgg_views_send_header_x_frame_options()` from this hook.

output, page In `elgg_view_page()`, this filters the output return value.

output:before, layout In `elgg_view_layout()`, filters `$params` before it's passed to the layout view.

output:after, layout In `elgg_view_layout()`, filters the return value of the layout view.

parameters, menu:<menu_name> Triggered by `elgg_view_menu()`. Used to change menu variables (like sort order) before rendering.

register, menu:<nombre del menú> Filters the initial list of menu items pulled from configuration, before the menu has been split into sections. Triggered by `elgg_view_menu()` and `elgg()->menus->getMenu()`.

prepare, menu:<nombre del menú> Filters the array of menu sections before they're displayed. Each section is a string key mapping to an area of menu items. This is a good hook to sort, add, remove, and modify menu items. Triggered by `elgg_view_menu()` and `elgg()->menus->prepareMenu()`.

creating, river The options for `elgg_create_river_item` are filtered through this hook. You may alter values or return `false` to cancel the item creation.

simplecache:generate, <view> Triggered when generating the cached content of a view.

prepare, breadcrumbs In elgg_get_breadcrumbs(), this filters the registered breadcrumbs before returning them, allowing a plugin to alter breadcrumb strategy site-wide.

add, river

elgg.data, site Filters cached configuration data to pass to the client. *More info*

elgg.data, page Filters uncached, page-specific configuration data to pass to the client. *More info*

registration_url, site Filters site's registration URL. Can be used by plugins to attach invitation codes, referrer codes etc. to the registration URL. `$params` array contains an array of query elements added to the registration URL by the invoking script. The hook must return an absolute URL to the registration page.

login_url, site Filters site's login URL. `$params` array contains an array of query elements added to the login URL by the invoking script. The hook must return an absolute URL of the login page.

3.31.2 Ganchos de usuarios

usersettings:save, user Triggered in the aggregate action to save user settings. Return false prevent sticky forms from being cleared.

access:collections:write, user Filters an array of access permissions that the user `$params['user_id']` is allowed to save content with. Permissions returned are of the form (id => "Human Readable Name").

registeruser:validate:username, all Return boolean for if the string in `$params['username']` is valid for a username.

registeruser:validate:password, all Return boolean for if the string in `$params['password']` is valid for a password.

registeruser:validate:email, all Return boolean for if the string in `$params['email']` is valid for an email address.

register, user Triggered by the `register` action after the user registers. Return `false` to delete the user. Note the function `register_user` does *not* trigger this hook.

login:forward, user Filters the URL to which the user will be forwarded after login.

find_active_users, system Return the number of active users.

status, user Triggered by The Wire when adding a post.

username:character_blacklist, user Filters the string of blacklisted characters used to validate username during registration. The return value should be a string consisting of the disallowed characters. The default string can be found from `$params['blacklist']`.

3.31.3 Ganchos de objetos

comments, <tipo de entidad> Triggered in `elgg_view_comments()`. If returning content, this overrides the `page/elements/comments` view.

comments:count, <tipo de entidad> Return the number of comments on `$params['entity']`.

likes:count, <tipo de entidad> Return the number of likes for `$params['entity']`.

3.31.4 Ganchos de acciones

action, <acción> Triggered before executing action scripts. Return false to abort action.

action_gatekeeper:permissions:check, all Triggered after a CSRF token is validated. Return false to prevent validation.

action_gatekeeper:upload_exceeded_msg, all Triggered when a POST exceeds the max size allowed by the server. Return an error message to display.

forward, <motivo> Filter the URL to forward a user to when `forward($url, $reason)` is called.

response, action:<action> Filter an instance of `\Elgg\Http\ResponseBuilder` before it is sent to the client. This hook can be used to modify response content, status code, forward URL, or set additional response headers. Note that the `<action>` value is parsed from the request URL, therefore you may not be able to filter the responses of *action()* calls if they are nested within the another action script file.

3.31.5 AJAX

ajax_response, * When the `elgg/Ajax` AMD module is used, this hook gives access to the response object (`\Elgg\Services\AjaxResponse`) so it can be altered/extended. The hook type depends on the method call:

elgg/Ajax method	plugin hook type
action()	action:<action_name>
path()	path:<url_path>
view()	view:<view_name>
form()	form:<action_name>

output, ajax This filters the JSON output wrapper returned to the legacy ajax API (`elgg.ajax`, `elgg.action`, etc.). Plugins can alter the output, forward URL, system messages, and errors. For the `elgg/Ajax` AMD module, use the `ajax_response` hook documented above.

3.31.6 Ganchos de permisos

container_logic_check, <entity_type> Triggered by `ElggEntity:canWriteToContainer()` before triggering `permissions_check` and `container_permissions_check` hooks. Unlike permissions hooks, logic check can be used to prevent certain entity types from being contained by other entity types, e.g. discussion replies should only be contained by discussions. This hook can also be used to apply status logic, e.g. do disallow new replies for closed discussions.

The handler should return `false` to prevent an entity from containing another entity. The default value passed to the hook is `null`, so the handler can check if another hook has modified the value by checking if return value is set. Should this hook return `false`, `container_permissions_check` and `permissions_check` hooks will not be triggered.

The `$params` array will contain:

- `container` - An entity that will be used as a container
- `user` - User who will own the entity to be written to container
- `subtype` - Subtype of the entity to be written to container (entity type is assumed from hook type)

container_permissions_check, <tipo de entidad> Return boolean for if the user `$params['user']` can use the entity `$params['container']` as a container for an entity of `<entity_type>` and subtype `$params['subtype']`.

In the rare case where an entity is created with neither the `container_guid` nor the `owner_guid` matching the logged in user, this hook is called *twice*, and in the first call `$params['container']` will be the *owner*, not the entity's real container.

The `$params` array will contain:

- `container` - An entity that will be used as a container
- `user` - User who will own the entity to be written to container
- `subtype` - Subtype of the entity to be written to container (entity type is assumed from hook type)

permissions_check, <tipo de entidad> Return boolean for if the user `$params['user']` can edit the entity `$params['entity']`.

permissions_check:delete, <entity_type> Return boolean for if the user `$params['user']` can delete the entity `$params['entity']`. Defaults to `$entity->canEdit()`.

permissions_check:delete, river Return boolean for if the user `$params['user']` can delete the river item `$params['item']`. Defaults to `true` for admins and `false` for other users.

Nota: This check is not performed when using the deprecated `elgg_delete_river()`.

---

permissions_check, widget_layout Return boolean for if `$params['user']` can edit the widgets in the context passed as `$params['context']` and with a page owner of `$params['page_owner']`.

permissions_check:metadata, <tipo de entidad> Return boolean for if the user `$params['user']` can edit the metadata `$params['metadata']` on the entity `$params['entity']`.

permissions_check:comment, <tipo de entidad> Return boolean for if the user `$params['user']` can comment on the entity `$params['entity']`.

permissions_check:annotate:<annotation_name>, <entity_type> Return boolean for if the user `$params['user']` can create an annotation `<annotation_name>` on the entity `$params['entity']`. If logged in, the default is true.

Nota: This is called before the more general `permissions_check:annotate` hook, and its return value is that hook's initial value.

---

permissions_check:annotate, <entity_type> Return boolean for if the user `$params['user']` can create an annotation `$params['annotation_name']` on the entity `$params['entity']`. if logged in, the default is true.

Advertencia: This is functions differently than the `permissions_check:metadata` hook by passing the annotation name instead of the metadata object.

permissions_check:annotation Return boolean for if the user in `$params['user']` can edit the annotation `$params['annotation']` on the entity `$params['entity']`. The user can be null.

fail, auth Return the failure message if authentication failed. An array of previous PAM failure methods is passed as `$params`.

api_key, use Triggered by `api_auth_key()`. Returning false prevents the key from being authenticated.

access:collections:read, user Filters an array of access IDs that the user `$params['user_id']` can see.

> **Advertencia:** The handler needs to either not use parts of the API that use the access system (triggering the hook again) or to ignore the second call. Otherwise, an infinite loop will be created.

access:collections:write, user Filters an array of access IDs that the user `$params['user_id']` can write to. In get_write_access_array(), this hook filters the return value, so it can be used to alter the available options in the input/access view. For core plugins, the value «input_params» has the keys «entity» (ElggEntity|false), «entity_type» (string), «entity_subtype» (string), «container_guid» (int) are provided. An empty entity value generally means the form is to create a new object.

> **Advertencia:** The handler needs to either not use parts of the API that use the access system (triggering the hook again) or to ignore the second call. Otherwise, an infinite loop will be created.

access:collections:addcollection, collection Triggered after an access collection `$params['collection_id']` is created.

access:collections:deletecollection, collection Triggered before an access collection `$params['collection_id']` is deleted. Return false to prevent deletion.

access:collections:add_user, collection Triggered before adding user `$params['user_id']` to collection `$params['collection_id']`. Return false to prevent adding.

access:collections:remove_user, collection Triggered before removing user `$params['user_id']` to collection `$params['collection_id']`. Return false to prevent removal.

get_sql, access Filters the SQL clauses used in `_elgg_get_access_where_sql()`.

gatekeeper, <entity_type>:<entity_subtype> Filters the result of `elgg_entity_gatekeeper()` to prevent access to an entity that user would otherwise have access to. A handler should return false to deny access to an entity.

3.31.7 Notificaciones

These hooks are listed chronologically in the lifetime of the notification event. Note that not all hooks apply to instant notifications.

enqueue, notification Can be used to prevent a notification event from sending **subscription** notifications. Hook handler must return `false` to prevent a subscription notification event from being enqueued.

`$params` array includes:

- `object` - object of the notification event
- `action` - action that triggered the notification event. E.g. corresponds to `publish` when `elgg_trigger_event('publish', 'object', $object)` is called

get, subscriptions

Filters subscribers of the notification event. Applies to **subscriptions** and **instant** notifications. In case of a subscription event, by default, the subscribers list consists of the users subscribed to the container entity of the event object. In case of an instant notification event, the subscribers list consists of the users passed as recipients to `notify_user()`

IMPORTANT Always validate the notification event, object and/or action types before adding any new recipients to ensure that you do not accidentally dispatch notifications to unintended recipients. Consider a situation, where a mentions plugin sends out an instant notification to a mentioned user - any hook acting on a subject or an object without validating an event or action type (e.g. including an owner of the original wire thread) might end up sending notifications to wrong users.

`$params` array includes:

- `event` - `\Elgg\Notifications\NotificationEvent` instance that describes the notification event
- `origin` - `subscriptions_service` or `instant_notifications`
- `methods_override` - delivery method preference for instant notifications

Handlers must return an array in the form:

```
array(
        <user guid> => array('sms'),
        <user_guid2> => array('email', 'sms', 'ajax')
);
```

send:before, notifications Triggered before the notification event queue is processed. Can be used to terminate the notification event. Applies to **subscriptions** and **instant** notifications.

`$params` array includes:

- `event` - `\Elgg\Notifications\NotificationEvent` instance that describes the notification event
- `subscriptions` - a list of subscriptions. See `'get', 'subscriptions'` hook for details

prepare, notification A high level hook that can be used to alter an instance of `\Elgg\Notifications\Notification` before it is sent to the user. Applies to **subscriptions** and **instant** notifications. This hook is triggered before a more granular `'prepare', 'notification:<action>:<entity_type>:<entity_subtype>'` and after `'send:before', 'notifications`. Hook handler should return an altered notification object.

`$params` may vary based on the notification type and may include:

- `event` - `\Elgg\Notifications\NotificationEvent` instance that describes the notification event
- `object` - object of the notification `event`. Can be `null` for instant notifications
- `action` - action that triggered the notification `event`. May default to `notify_user` for instant notifications
- `method` - delivery method (e.g. `email`, `site`)
- `sender` - sender
- `recipient` - recipient
- `language` - language of the notification (recipient's language)
- `origin` - `subscriptions_service` or `instant_notifications`

prepare, notification:<action>:<entity_type>:<entity_type> A granular hook that can be used to filter a notification `\Elgg\Notifications\Notification` before it is sent to the user. Applies to **subscriptions** and **instant** notifications. In case of instant notifications that have not received an object, the hook will be called as `'prepare', 'notification:<action>'`. In case of instant notifications that have not received an action name, it will default to `notify_user`.

`$params` include:

- `event` - `\Elgg\Notifications\NotificationEvent` instance that describes the notification event
- `object` - object of the notification `event`. Can be `null` for instant notifications
- `action` - action that triggered the notification `event`. May default to `notify_user` for instant notifications
- `method` - delivery method (e.g. `email`, `site`)
- `sender` - sender
- `recipient` - recipient
- `language` - language of the notification (recipient's language)
- `origin` - `subscriptions_service` or `instant_notifications`

format, notification:<method> This hook can be used to format a notification before it is passed to the `'send', 'notification:<method>'` hook. Applies to **subscriptions** and **instant** notifications. The hook handler should return an instance of `\Elgg\Notifications\Notification`. The hook does not receive any `$params`. Some of the use cases include:

- Strip tags from notification title and body for plaintext email notifications
- Inline HTML styles for HTML email notifications
- Wrap notification in a template, add signature etc.

send, notification:<method> Delivers a notification. Applies to **subscriptions** and **instant** notifications. The handler must return `true` or `false` indicating the success of the delivery.

`$params` array includes:

- `notification` - a notification object `\Elgg\Notifications\Notification`

email, system Triggered by `elgg_send_email()`. Applies to **subscriptions** and **instant** notifications with `email` method. This hook can be used to alter email parameters (subject, body, headers etc) - the handler should return an array of altered parameters. This hook can also be used to implement a custom email transport (in place of Elgg's default plaintext `\Zend\Mail\Transport\Sendmail`) - the handler must return `true` or `false` to indicate whether the email was sent using a custom transport.

`$params` contains:

- `to` - email address or string in the form `Name <name@example.org>` of the recipient
- `from` - email address or string in the form `Name <name@example.org>` of the sender
- `subject` - subject line of the email
- `body` - body of the email
- `headers` - an array of headers
- `params` - other parameters inherited from the notification object or passed directly to `elgg_send_email()`

send:after, notifications Triggered after all notifications in the queue for the notifications event have been processed. Applies to **subscriptions** and **instant** notifications.

`$params` array includes:

- `event` - `\Elgg\Notifications\NotificationEvent` instance that describes the notification event
- `subscriptions` - a list of subscriptions. See `'get', 'subscriptions'` hook for details
- `deliveries` - a matrix of delivery statuses by user for each delivery method

3.31.8 Encaminamiento

route, <identifier> Allows applying logic or returning a response before the page handler is called. See *Encaminamiento* for details. Note that plugins using this hook to rewrite paths, will not be able to filter the response object by its final path and should either switch to `route:rewrite, <identifier>` hook or use `response, path:<path>` hook for the original path.

route:rewrite, <identifier> Allows altering the site-relative URL path. See *Encaminamiento* for details.

response, path:<path> Filter an instance of `\Elgg\Http\ResponseBuilder` before it is sent to the client. This hook type will only be used if the path did not start with «action/» or «ajax/». This hook can be used to modify response content, status code, forward URL, or set additional response headers. Note that the `<path>` value is parsed from the request URL, therefore plugins using the `route` hook should use the original `<path>` to filter the response, or switch to using the `route:rewrite` hook.

ajax_response, path:<path> Filters ajax responses before they're sent back to the `elgg/Ajax` module. This hook type will only be used if the path did not start with «action/» or «ajax/».

3.31.9 Vistas

view_vars, <view_name> Filters the `$vars` array passed to the view

view, <nombre de la vista> Filters the returned content of the view

layout, page In `elgg_view_layout()`, filters the layout name

shell, page In `elgg_view_page()`, filters the page shell name

head, page In `elgg_view_page()`, filters `$vars['head']` Return value contains an array with `title`, `metas` and `links` keys, where `metas` is an array of elements to be formatted as `<meta>` head tags, and `links` is an array of elements to be formatted as `<link>` head tags. Each meta and link element contains a set of key/value pairs that are formatted into html tag attributes, e.g.

```
return [
   'title' => 'Current page title',
   'metas' => [
      'viewport' => [
         'name' => 'viewport',
         'content' => 'width=device-width',
          ]
   ],
   'links' => [
      'rss' => [
         'rel' => 'alternative',
         'type' => 'application/rss+xml',
         'title' => 'RSS',
```

(continué en la próxima página)

(proviene de la página anterior)

```
            'href' => elgg_format_url($url),
        ],
        'icon-16' => [
            'rel' => 'icon',
            'sizes' => '16x16',
            'type' => 'image/png',
                    'href' => elgg_get_simplecache_url('favicon-16.png'),
        ],
    ],
];
```

ajax_response, view:<view> Filters `ajax/view/` responses before they're sent back to the `elgg/Ajax` module.

ajax_response, form:<action> Filters `ajax/form/` responses before they're sent back to the `elgg/Ajax` module.

response, view:<view_name> Filter an instance of `\Elgg\Http\ResponseBuilder` before it is sent to the client. Applies to request to `/ajax/view/<view_name>`. This hook can be used to modify response content, status code, forward URL, or set additional response headers.

response, form:<form_name> Filter an instance of `\Elgg\Http\ResponseBuilder` before it is sent to the client. Applies to request to `/ajax/form/<form_name>`. This hook can be used to modify response content, status code, forward URL, or set additional response headers.

table_columns:call, <name> When the method `elgg()->table_columns->$name()` is called, this hook is called to allow plugins to override or provide an implementation. Handlers receive the method arguments via `$params['arguments']` and should return an instance of `Elgg\Views\TableColumn` if they wish to specify the column directly.

3.31.10 Files

mime_type, file Return the mimetype for the filename `$params['filename']` with original filename `$params['original_filename']` and with the default detected mimetype of `$params['default']`.

simple_type, file In `elgg_get_file_simple_type()`, filters the return value. The hook uses `$params['mime_type']` (e.g. `application/pdf` or `image/jpeg`) and determines an overall category like `document` or `image`. The bundled file plugin and other-third party plugins usually store `simpletype` metadata on file entities and make use of it when serving icons and constructing `ege*` filters and menus.

upload, file Allows plugins to implement custom logic for moving an uploaded file into an instance of `ElggFile`. The handler must return `true` to indicate that the uploaded file was moved. The handler must return `false` to indicate that the uploaded file could not be moved. Other returns will indicate that `ElggFile::acceptUploadedFile` should proceed with the default upload logic.

`$params` array includes:

- `file` - instance of `ElggFile` to write to
- `upload` - instance of Symfony's `UploadedFile`

3.31.11 Otro

config, comments_per_page Filters the number of comments displayed per page. Default is 25.

default, access In get_default_access(), this hook filters the return value, so it can be used to alter the default value in the input/access view. For core plugins, the value «input_params» has the keys «entity» (ElggEntity|false), «entity_type» (string), «entity_subtype» (string), «container_guid» (int) are provided. An empty entity value generally means the form is to create a new object.

entity:icon:sizes, <entity_type> Triggered by `elgg_get_icon_sizes()` and sets entity type/subtype specific icon sizes. `entity_subtype` will be passed with the `$params` array to the callback.

entity:<icon_type>:sizes, <entity_type> Allows filtering sizes for custom icon types, see `entity:icon:sizes, <entity_type>`.

The hook must return an associative array where keys are the names of the icon sizes (e.g. «large»), and the values are arrays with the following keys:

- `w` - Width of the image in pixels
- `h` - Height of the image in pixels
- `square` - Should the aspect ratio be a square (true/false)
- `upscale` - Should the image be upscaled in case it is smaller than the given width and height (true/false)

If the configuration array for an image size is empty, the image will be saved as an exact copy of the source without resizing or cropping.

Ejemplo:

```
return [
        'small' => [
                'w' => 60,
                'h' => 60,
                'square' => true,
                'upscale' => true,
        ],
        'large' => [
                'w' => 600,
                'h' => 600,
                'upscale' => false,
        ],
        'original' => [],
];
```

entity:icon:url, <tipo de entidad> Triggered when entity icon URL is requested, see *entity icons*. Callback should return URL for the icon of size `$params['size']` for the entity `$params['entity']`. Following parameters are available through the `$params` array:

entity Entity for which icon url is requested.

viewtype The type of *view* e.g. `'default'` or `'json'`.

size Size requested, see *entity icons* for possible values.

Example on how one could default to a Gravatar icon for users that have not yet uploaded an avatar:

```
// Priority 600 so that handler is triggered after avatar handler
elgg_register_plugin_hook_handler('entity:icon:url', 'user', 'gravatar_icon_handler',␣
↪600);

/**
 * Default to icon from gravatar for users without avatar.
 */
function gravatar_icon_handler($hook, $type, $url, $params) {
```

(continué en la próxima página)

(proviene de la página anterior)

```
        // Allow users to upload avatars
        if ($params['entity']->icontime) {
                return $url;
        }

        // Generate gravatar hash for user email
        $hash = md5(strtolower(trim($params['entity']->email)));

        // Default icon size
        $size = '150x150';

        // Use configured size if possible
        $config = elgg_get_icon_sizes('user');
        $key = $params['size'];
        if (isset($config[$key])) {
                $size = $config[$key]['w'] . 'x' . $config[$key]['h'];
        }

        // Produce URL used to retrieve icon
        return "http://www.gravatar.com/avatar/$hash?s=$size";
}
```

entity:<icon_type>:url, <entity_type> Allows filtering URLs for custom icon types, see `entity:icon:url, <entity_type>`

entity:icon:file, <entity_type> Triggered by `ElggEntity::getIcon()` and allows plugins to provide an alternative `ElggIcon` object that points to a custom location of the icon on filestore. The handler must return an instance of `ElggIcon` or an exception will be thrown.

entity:<icon_type>:file, <entity_type> Allows filtering icon file object for custom icon types, see `entity:icon:file, <entity_type>`

entity:<icon_type>:prepare, <entity_type> Triggered by `ElggEntity::saveIcon*()` methods and can be used to prepare an image from uploaded/linked file. This hook can be used to e.g. rotate the image before it is resized/cropped, or it can be used to extract an image frame if the uploaded file is a video. The handler must return an instance of `ElggFile` with a *simpletype* that resolves to *image*. The `$return` value passed to the hook is an instance of `ElggFile` that points to a temporary copy of the uploaded/linked file.

The `$params` array contains:

- `entity` - entity that owns the icons
- `file` - original input file before it has been modified by other hooks

entity:<icon_type>:save, <entity_type> Triggered by `ElggEntity::saveIcon*()` methods and can be used to apply custom image manipulation logic to resizing/cropping icons. The handler must return `true` to prevent the core APIs from resizing/cropping icons. The `$params` array contains:

- `entity` - entity that owns the icons
- `file` - `ElggFile` object that points to the image file to be used as source for icons
- `x1`, `y1`, `x2`, `y2` - cropping coordinates

entity:<icon_type>:saved, <entity_type> Triggered by `ElggEntity::saveIcon*()` methods once icons have been created. This hook can be used by plugins to create river items, update cropping coordinates for custom icon types etc. The handler can access the created icons using `ElggEntity::getIcon()`. The `$params` array contains:

- `entity` - entity that owns the icons

- `x1`, `y1`, `x2`, `y2` - cropping coordinates

entity:<icon_type>:delete, <entity_type> Triggered by `ElggEntity::deleteIcon()` method and can be used for clean up operations. This hook is triggered before the icons are deleted. The handler can return `false` to prevent icons from being deleted. The `$params` array contains:

- `entity` - entity that owns the icons

entity:url, <entity_type> Return the URL for the entity `$params['entity']`. Note: Generally it is better to override the `getUrl()` method of ElggEntity. This hook should be used when it's not possible to subclass (like if you want to extend a bundled plugin without overriding many views).

to:object, <entity_type|metadata|annotation|relationship|river_item> Converts the entity `$params['entity']` to a StdClass object. This is used mostly for exporting entity properties for portable data formats like JSON and XML.

extender:url, <annotation|metadata> Return the URL for the annotation or metadatum `$params['extender']`.

file:icon:url, override Override a file icon URL.

is_member, group Return boolean for if the user `$params['user']` is a member of the group `$params['group']`.

entity:annotate, <tipo de entidad> Triggered in `elgg_view_entity_annotations()`, which is called by `elgg_view_entity()`. Can be used to add annotations to all full entity views.

usersetting, plugin Filter user settings for plugins. `$params` contains:

- `user` - An ElggUser instance
- `plugin` - An ElggPlugin instance
- `plugin_id` - The plugin ID
- `name` - The name of the setting
- `value` - The value to set

setting, plugin Filter plugin settings. `$params` contains:

- `plugin` - An ElggPlugin instance
- `plugin_id` - The plugin ID
- `name` - The name of the setting
- `value` - The value to set

relationship:url, <relationship_name> Filter the URL for the relationship object `$params['relationship']`.

profile:fields, group Filter an array of profile fields. The result should be returned as an array in the format `name => input view name`. For example:

```
array(
        'about' => 'longtext'
);
```

profile:fields, profile Filter an array of profile fields. The result should be returned as an array in the format `name => input view name`. For example:

```
array(
        'about' => 'longtext'
);
```

widget_settings, <manejador de artilugio> Triggered when saving a widget settings `$params['params']` for widget `$params['widget']`. If handling saving the settings, the handler should return true to prevent the default code from running.

handlers, widgets Triggered when a list of available widgets is needed. Plugins can conditionally add or remove widgets from this list or modify attributes of existing widgets like `context` or `multiple`.

get_list, default_widgets Filters a list of default widgets to add for newly registered users. The list is an array of arrays in the format:

```
array(
        'event' => $event,
        'entity_type' => $entity_type,
        'entity_subtype' => $entity_subtype,
        'widget_context' => $widget_context
)
```

public_pages, walled_garden Filter the URLs that are can be seen by logged out users if Walled Garden is enabled. `$value` is an array of regex strings that will allow access if matched.

volatile, metadata Triggered when exporting an entity through the export handler. This is rare. This allows handler to handle any volatile (non-persisted) metadata on the entity. It's preferred to use the `to:object, <type>` hook.

maintenance:allow, url

Return boolean if the URL `$params['current_url']` and the path `$params['current_path']` is allowed during maintenance mode.

robots.txt, site Filter the robots.txt values for `$params['site']`.

config, amd Filter the AMD config for the requirejs library.

3.31.12 Complementos

Incrustado

embed_get_items, <sección activa>

embed_get_sections, all

embed_get_upload_sections, all

Groups

profile_buttons, group Filters buttons (`ElggMenuItem` instances) to be registered in the title menu of the group profile page

tool_options, group Use this hook to influence the available group tool options

HTMLawed

allowed_styles, htmlawed Filter the HTMLawed allowed style array.

config, htmlawed Filter the HTMLawed config array.

Likes

likes:is_likable, <type>:<subtype> This is called to set the default permissions for whether to display/allow likes on an entity of type `<type>` and subtype `<subtype>`.

> **Nota:** The callback `'Elgg\Values::getTrue'` is a useful handler for this hook.

Miembros

members:list, <segmento de página> To handle the page `/members/$page_segment`, register for this hook and return the HTML of the list.

members:config, tabs Este gancho se usa para crear un vector de pestañas a pasarle a la vista de navegación o pestañas para los miembros de las páginas.

API de Twitter

authorize, twitter_api Triggered when a user is authorizes Twitter for a login. `$params['token']` contains the Twitter authorization token.

Contenido denunciado

reportedcontent:add, system Triggered after adding the reported content object `$params['report']`. Return false to delete report.

reportedcontent:archive, system Triggered before archiving the reported content object `$params['report']`. Return false to prevent archiving.

reportedcontent:delete, system Triggered before deleting the reported content object `$params['report']`. Return false to prevent deleting.

Buscar

search, <tipo>:<subtipo> Filter more granular search results than searching by type alone. Must return an array with `count` as the total count of results and `entities` an array of ElggUser entities.

search, tags

search, <tipo> Filter the search for entities for type `$type`. Must return an array with `count` as the total count of results and `entities` an array of ElggUser entities.

search_types, get_types Filter an array of search types. This allows plugins to add custom types that don't correspond directly to entities.

search_types, get_queries Antes de una búsqueda, esto filtra los tipos que se indiquen. Esto puede usarse para reordenar los resultados que se muestran en una búsqueda.

Web Services

rest, init Triggered by the web services rest handler. Plugins can set up their own authentication handlers, then return `true` to prevent the default handlers from being registered.

rest:output, <method_name> Filter the result (and subsequently the output) of the API method

CAPÍTULO 4

Tutorials

Walk through all the required steps in order to customize Elgg.

The instructions are detailed enough that you don't need much previous experience with Elgg.

4.1 Hello world

This tutorial shows you how to create a new plugin that consists of a new page with the text «Hello world» on it.

Before anything else, you need to *install Elgg*.

In this tutorial we will pretend your site's URL is `https://elgg.example.com`.

First, create a directory that will contain the plugin's files. It should be located under the `mod/` directory which is located in your Elgg installation directory. So in this case, create `mod/hello/`.

4.1.1 Manifest file

Elgg requires that your plugin has a manifest file that contains information about the plugin. Therefore, in the directory you just created, create a file called `manifest.xml` and copy this code into it:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
    <name>Hello world</name>
    <id>hello</id>
    <author>Your Name Here</author>
    <version>0.1</version>
    <description>Hello world, testing.</description>
    <requires>
        <type>elgg_release</type>
        <version>2.0</version>
    </requires>
</plugin_manifest>
```

This is the minimum amount of information in a manifest file:

- <name> is the display name of the plugin
- <id> must be the same as the directory you just created
- <requires> must include which version of Elgg your plugin requires
- <author>, <version> and <description> should have some appropriate values but can be filled freely

4.1.2 Initializer

Next, create start.php in the mod/hello/ directory and copy this code into it:

```
<?php

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

function hello_world_init() {

}
```

The above code tells Elgg that it should call the function hello_world_init() once the Elgg core system is initiated.

4.1.3 Registering a page handler

The next step is to register a page handler which has the purpose of handling request that users make to the URL https://elgg.example.com/hello.

Update start.php to look like this:

```
<?php

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

function hello_world_init() {
    elgg_register_page_handler('hello', 'hello_world_page_handler');
}

function hello_world_page_handler() {
    echo elgg_view_resource('hello');
}
```

The call to elgg_register_page_handler() tells Elgg that it should call the function hello_world_page_handler() when a user navigates to https://elgg.example.com/hello/*.

The hello_world_page_handler() passes off rendering the actual page to a view file called hello.php.

4.1.4 View file

Create mod/hello/views/default/resources/hello.php with this content:

```
<?php

$params = array(
    'title' => 'Hello world!',
    'content' => 'My first page!',
    'filter' => '',
);

$body = elgg_view_layout('content', $params);

echo elgg_view_page('Hello', $body);
```

The code creates an array of parameters to be given to the `elgg_view_layout()` function, including:

- The title of the page
- The contents of the page
- Filter which is left empty because there's currently nothing to filter

This creates the basic layout for the page. The layout is then run through `elgg_view_page()` which assembles and outputs the full page.

4.1.5 Last step

Finally, activate the plugin through your Elgg administrator page: `https://elgg.example.com/admin/plugins` (the new plugin appears at the bottom).

You can now go to the address `https://elgg.example.com/hello/` and you should see your new page!

4.2 Customizing the Home Page

To override the homepage, just override Elgg's `resources/index` view by creating a file at `/views/default/resources/index.php`.

Any output from this view will become your new homepage.

You can take a similar approach with any other page in Elgg or official plugins.

4.3 Building a Blog Plugin

This tutorial will teach you how to create a simple blog plugin. The basic functions of the blog will be creating posts, saving them and viewing them. The plugin duplicates features that are found in the bundled `blog` plugin. You can disable the bundled `blog` plugin if you wish, but it is not necessary since the features do not conflict each other.

Contents

Prerequisites:

- *Install Elgg*

4.3.1 Create the plugin's directory and manifest file

First, choose a simple and descriptive name for your plugin. In this tutorial, the name will be `my_blog`. Then, create a directory for your plugin in the `/mod/` directory found in your Elgg installation directory. Other plugins are also located in `/mod/`. In this case, the name of the directory should be `/mod/my_blog/`. This directory is the root of your plugin and all the files that you create for the new plugin will go somewhere under it.

Next, in the root of the plugin, create the plugin's manifest file, `manifest.xml`:

```
<?xml version="1.0" encoding="UTF-8"?>
<plugin_manifest xmlns="http://www.elgg.org/plugin_manifest/1.8">
    <name>My Blog</name>
    <id>my_blog</id>
    <author>Your Name Here</author>
    <version>0.1</version>
    <description>Adds blogging capabilities.</description>
    <requires>
        <type>elgg_release</type>
        <version>2.0</version>
    </requires>
</plugin_manifest>
```

See *Plugins* for more information about the manifest file.

4.3.2 Create the form for creating a new blog post

Create a file at `/mod/my_blog/views/default/forms/my_blog/save.php` that contains the form body. The form should have input fields for the title, body and tags of the my_blog post. It does not need form tag markup.

```
echo elgg_view_field([
    '#type' => 'text',
    '#label' => elgg_echo('title'),
    'name' => 'title',
    'required' => true,
]);

echo elgg_view_field([
    '#type' => 'longtext',
    '#label' => elgg_echo('body'),
    'name' => 'body',
    'required' => true,
```

(continué en la próxima página)

(proviene de la página anterior)

```
]);

echo elgg_view_field([
    '#type' => 'tags',
    '#label' => elgg_echo('tags'),
    '#help' => elgg_echo('tags:help'),
    'name' => 'tags',
]);

$submit = elgg_view_field(array(
    '#type' => 'submit',
    '#class' => 'elgg-foot',
    'value' => elgg_echo('save'),
));
elgg_set_form_footer($submit);
```

Notice how the form is calling `elgg_view_field()` to render inputs. This helper function maintains consistency in field markup, and is used as a shortcut for rendering field elements, such as label, help text, and input. See *Formularios y acciones*.

You can see a complete list of input views in the `/vendor/elgg/elgg/views/default/input/` directory.

It is recommended that you make your plugin translatable by using `elgg_echo()` whenever there is a string of text that will be shown to the user. Read more at *Internationalization*.

4.3.3 Create a page for composing the blogs

Create the file `/mod/my_blog/views/default/resources/my_blog/add.php`. This page will view the form you created in the above section.

```
<?php
// make sure only logged in users can see this page
gatekeeper();

// set the title
$title = "Create a new my_blog post";

// start building the main column of the page
$content = elgg_view_title($title);

// add the form to the main column
$content .= elgg_view_form("my_blog/save");

// optionally, add the content for the sidebar
$sidebar = "";

// layout the page
$body = elgg_view_layout('one_sidebar', array(
   'content' => $content,
   'sidebar' => $sidebar
));

// draw the page, including the HTML wrapper and basic page layout
echo elgg_view_page($title, $body);
```

The function `elgg_view_form("my_blog/save")` views the form that you created in the previous section. It

also automatically wraps the form with a `<form>` tag and the necessary attributes as well as anti-csrf tokens.

The form's action will be `"<?= elgg_get_site_url() ?>action/my_blog/save"`.

4.3.4 Create the action file for saving the blog post

The action file will save the my_blog post to the database. Create the file `/mod/my_blog/actions/my_blog/save.php`:

```
<?php
// get the form inputs
$title = get_input('title');
$body = get_input('body');
$tags = string_to_tag_array(get_input('tags'));

// create a new my_blog object and put the content in it
$blog = new ElggObject();
$blog->title = $title;
$blog->description = $body;
$blog->tags = $tags;

// the object can and should have a subtype
$blog->subtype = 'my_blog';

// for now, make all my_blog posts public
$blog->access_id = ACCESS_PUBLIC;

// owner is logged in user
$blog->owner_guid = elgg_get_logged_in_user_guid();

// save to database and get id of the new my_blog
$blog_guid = $blog->save();

// if the my_blog was saved, we want to display the new post
// otherwise, we want to register an error and forward back to the form
if ($blog_guid) {
   system_message("Your blog post was saved.");
   forward($blog->getURL());
} else {
   register_error("The blog post could not be saved.");
   forward(REFERER); // REFERER is a global variable that defines the previous page
}
```

As you can see in the above code, Elgg objects have several fields built into them. The title of the my_blog post is stored in the `title` field while the body is stored in the `description` field. There is also a field for tags which are stored as metadata.

Objects in Elgg are a subclass of something called an «entity». Users, sites, and groups are also subclasses of entity. An entity's subtype allows granular control for listing and displaying, which is why every entity should have a subtype. In this tutorial, the subtype «`my_blog`» identifies a my_blog post, but any alphanumeric string can be a valid subtype. When picking subtypes, be sure to pick ones that make sense for your plugin.

The `getURL` method fetches the URL of the new post. It is recommended that you override this method. The overriding will be done in the `start.php` file.

4.3.5 Create start.php

The `/mod/my_blog/start.php` file needs to register the save action you created earlier, register a page handler and override the URL generation.

```
<?php

// register an initializer
elgg_register_event_handler('init', 'system', 'my_blog_init');

function my_blog_init() {
    // register the save action
    elgg_register_action("my_blog/save", __DIR__ . "/actions/my_blog/save.php");

    // register the page handler
    elgg_register_page_handler('my_blog', 'my_blog_page_handler');

    // register a hook handler to override urls
    elgg_register_plugin_hook_handler('entity:url', 'object', 'my_blog_set_url');
}
```

Registering the save action will make it available as `/action/my_blog/save`. By default, all actions are available only to logged in users. If you want to make an action available to only admins or open it up to unauthenticated users, you can pass "admin" or "public" as the third parameter of `elgg_register_action`.

The URL overriding function will extract the ID of the given entity and use it to make a simple URL for the page that is supposed to view the entity. In this case the entity should of course be a my_blog post. Add this function to your `start.php` file:

```
function my_blog_set_url($hook, $type, $url, $params) {
    $entity = $params['entity'];
    if (elgg_instanceof($entity, 'object', 'my_blog')) {
        return "my_blog/view/{$entity->guid}";
    }
}
```

The page handler makes it possible to serve the page that generates the form and the page that views the post. The next section will show how to create the page that views the post. Add this function to your `start.php` file:

```
function my_blog_page_handler($segments) {
    if ($segments[0] == 'add') {
        echo elgg_view_resource('my_blog/add');
        return true;
    }

    else if ($segments[0] == 'view') {
        $resource_vars['guid'] = elgg_extract(1, $segments);
        echo elgg_view_resource('my_blog/view', $resource_vars);
        return true;
    }

    return false;
}
```

The `$segments` variable contains the different parts of the URL as separated by /.

Page handling functions need to return `true` or `false`. `true` means the page exists and has been handled by the page handler. `false` means that the page does not exist and the user will be forwarded to the site's 404 page (requested

page does not exist or not found). In this particular example, the URL must contain either `/my_blog/add` or `/my_blog/view/id` where id is a valid ID of an entity with the `my_blog` subtype. More information about page handling is at *Page handler*.

4.3.6 Create a page for viewing a blog post

To be able to view a my_blog post on its own page, you need to make a view page. Create the file `/mod/my_blog/views/default/resources/my_blog/view.php`:

```
<?php

// get the entity
$guid = elgg_extract('guid', $vars);
$my_blog = get_entity($guid);

// get the content of the post
$content = elgg_view_entity($my_blog, array('full_view' => true));

$params = array(
    'title' => $my_blog->title,
    'content' => $content,
    'filter' => '',
);

$body = elgg_view_layout('content', $params);

echo elgg_view_page($my_blog->title, $body);
```

This page has much in common with the `add.php` page. The biggest differences are that some information is extracted from the my_blog entity, and instead of viewing a form, the function `elgg_view_entity` is called. This function gives the information of the entity to something called the object view.

4.3.7 Create the object view

When `elgg_view_entity` is called or when my_blogs are viewed in a list for example, the object view will generate the appropriate content. Create the file `/mod/my_blog/views/default/object/my_blog.php`:

```
<?php

echo elgg_view('output/longtext', array('value' => $vars['entity']->description));
echo elgg_view('output/tags', array('tags' => $vars['entity']->tags));
```

As you can see in the previous section, each my_blog post is passed to the object view as `$vars['entity']`. (`$vars` is an array used in the views system to pass variables to a view.)

The last line takes the tags on the my_blog post and automatically displays them as a series of clickable links. Search is handled automatically.

(If you're wondering about the «`default`» in `/views/default/`, you can create alternative views. RSS, OpenDD, FOAF, mobile and others are all valid view types.)

4.3.8 Trying it out

Go to your Elgg site's administration page, list the plugins and activate the `my_blog` plugin.

The page to create a new my_blog post should now be accessible at `https://elgg.example.com/my_blog/add`, and after successfully saving the post, you should see it viewed on its own page.

4.3.9 Displaying a list of blog posts

Let's also create a page that lists my_blog entries that have been created.

Create `/mod/my_blog/views/default/resources/my_blog/all.php`:

```
<?php
$titlebar = "All Site My_Blogs";
$pagetitle = "List of all my_blogs";

$body = elgg_list_entities(array(
    'type' => 'object',
    'subtype' => 'my_blog',
));

$body = elgg_view_title($pagetitle) . elgg_view_layout('one_column', array('content'
↪=> $body));

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

The `elgg_list_entities` function grabs the latest my_blog posts and passes them to the object view file. Note that this function returns only the posts that the user can see, so access restrictions are handled transparently. The function (and its cousins) also transparently handles pagination and even creates an RSS feed for your my_blogs if you have defined that view.

The list function can also limit the my_blog posts to those of a specified user. For example, the function `elgg_get_logged_in_user_guid` grabs the Global Unique IDentifier (GUID) of the logged in user, and by giving that to `elgg_list_entities`, the list only displays the posts of the current user:

```
echo elgg_list_entities(array(
    'type' => 'object',
    'subtype' => 'my_blog',
    'owner_guid' => elgg_get_logged_in_user_guid()
));
```

Next, you will need to modify your my_blog page handler to grab the new page when the URL is set to `/my_blog/all`. Change the `my_blog_page_handler` function in `start.php` to look like this:

```
function my_blog_page_handler($segments) {
    switch ($segments[0]) {
        case 'add':
           echo elgg_view_resource('my_blog/add');
           break;

        case 'view':
            $resource_vars['guid'] = elgg_extract(1, $segments);
            echo elgg_view_resource('my_blog/view', $resource_vars);
            break;

        case 'all':
        default:
           echo elgg_view_resource('my_blog/all');
           break;
    }
```

(continué en la próxima página)

(proviene de la página anterior)

```
    return true;
}
```

Now, if the URL contains `/my_blog/all`, the user will see an «All Site My_Blogs» page. Because of the default case, the list of all my_blogs will also be shown if the URL is something invalid, like `/my_blog` or `/my_blog/xyz`.

You might also want to update the object view to handle different kinds of viewing, because otherwise the list of all my_blogs will also show the full content of all my_blogs. Change `/mod/my_blog/views/default/object/my_blog.php` to look like this:

```
<?php
$full = elgg_extract('full_view', $vars, FALSE);

// full view
if ($full) {
    echo elgg_view('output/longtext', array('value' => $vars['entity']->description));
    echo elgg_view('output/tags', array('tags' => $vars['entity']->tags));

// list view or short view
} else {
    // make a link out of the post's title
    echo elgg_view_title(
        elgg_view('output/url', array(
            'href' => $vars['entity']->getURL(),
            'text' => $vars['entity']->title,
            'is_trusted' => true
    )));
    echo elgg_view('output/tags', array('tags' => $vars['entity']->tags));
}
```

Now, if `full_view` is `true` (as it was pre-emptively set to be in *this section*), the object view will show the post's content and tags (the title is shown by `view.php`). Otherwise the object view will render just the title and tags of the post.

4.3.10 The end

There's much more that could be done, but hopefully this gives you a good idea of how to get started.

4.4 Integrating a Rich Text Editor

Build your own wysiwyg plugin.

Elgg is bundled with a plugin for CKEditor, and previously shipped with TinyMCE support. However, if you have a wysiwyg that you prefer, you could use this tutorial to help you build your own.

All forms in Elgg should try to use the provided input views located in `views/default/input`. If these views are used, then it is simple for plugin authors to replace a view, in this case `input/longtext`, with their wysiwyg.

4.4.1 Add the WYSIWYG library code

Now you need to upload TinyMCE into a directory in your plugin. We strongly encourage you to use `composer` to manage third-party dependencies, since it is so much easier to upgrade and maintain that way:

```
.. code:: shell
```

composer require bower-asset/tinymce

4.4.2 Tell Elgg when and how to load TinyMCE

Now that you have:

- created your start file
- intialized the plugin
- uploaded the wysiwyg code

It is time to tell Elgg how to apply TinyMCE to longtext fields.

We're going to do that by extending the input/longtext view and including some javascript. Create a view tinymce/longtext and add the following code:

```
<?php

    /**
     * Elgg long text input with the tinymce text editor intacts
     * Displays a long text input field
     *
     * @package ElggTinyMCE
     *
     *
     */

?>
<!-- include tinymce -->
<script language="javascript" type="text/javascript" src="<?php echo $vars['url']; ?>
↪mod/tinymce/tinymce/jscripts/tiny_mce/tiny_mce.js"></script>
<!-- intialise tinymce, you can find other configurations here http://wiki.moxiecode.
↪com/examples/tinymce/installation_example_01.php -->
<script language="javascript" type="text/javascript">
   tinyMCE.init({
    mode : "textareas",
    theme : "advanced",
    theme_advanced_buttons1 : "bold,italic,underline,separator,strikethrough,
↪justifyleft,justifycenter,justifyright, justifyfull,bullist,numlist,undo,redo,link,
↪unlink,image,blockquote,code",
    theme_advanced_buttons2 : "",
    theme_advanced_buttons3 : "",
    theme_advanced_toolbar_location : "top",
    theme_advanced_toolbar_align : "left",
    theme_advanced_statusbar_location : "bottom",
    theme_advanced_resizing : true,
    extended_valid_elements : "a[name|href|target|title|onclick],
↪img[class|src|border=0|alt|title|hspace|vspace|width|height|align|onmouseover|onmouseout|name],
↪
hr[class|width|size|noshade],font[face|size|color|style],span[class|align|style]"
});
</script>
```

Then, in your plugin's init function, extend the input/longtext view

```
function tinymce_init() {
    elgg_extend_view('input/longtext', 'tinymce/longtext');
}
```

That's it! Now every time someone uses input/longtext, TinyMCE will be loaded and applied to that textarea.

4.5 Basic Widget

Create a widget that will display "Hello, World!" and optionally any text the user wants.

In Elgg, widgets are those components that you can drag onto your profile or admin dashboard.

This tutorial assumes you are familiar with basic Elgg concepts such as:

- *Vistas*
- *Complementos*

You should review those if you get confused along the way.

Contents

4.5.1 Adding the widget view code

Elgg automatically scans particular directories under plugins looking for particular files. *Vistas* make it easy to add your display code or do other things like override default Elgg behavior. For now, we will just be adding the view code for your widget. Create a file at `/views/default/widgets/helloworld/content.php`. "helloworld" will be the name of your widget within the hello plugin. In this file add the code:

```
<?php

echo "Hello, world!";
```

This will add these words to the widget canvas when it is drawn. Elgg takes care of loading the widget.

4.5.2 Registering your widget

Elgg needs to be told explicitly that the plugin contains a widget so that it will scan the widget views directory. This is done by calling the elgg_register_widget_type() function. Edit `/start.php`. In it add these lines:

```
<?php

function hello_init() {
    elgg_register_widget_type([
        'id' => 'helloworld',
        'name' => 'Hello, world!',
        'description' => 'The "Hello, world!" widget',
```

(continué en la próxima página)

(proviene de la página anterior)

```
    ]);
}

elgg_register_event_handler('init', 'system', 'hello_init');
```

Now go to your profile page using a web browser and add the "hello, world" widget. It should display "Hello, world!".

Nota: For real widgets, it is always a good idea to support *Internacionalización*.

4.5.3 Allow user customization

Click on the edit link on the toolbar of the widget that you've created. You will notice that the only control it gives you by default is over access (over who can see the widget).

Suppose you want to allow the user to control what greeting is displayed in the widget. Just as Elgg automatically loads `content.php` when viewing a widget, it loads `edit.php` when a user attempts to edit a widget. Put the following code into `/views/default/widgets/helloworld/edit.php`:

```
<div>
    <label>Message:</label>
    <?php
        //This is an instance of the ElggWidget class that represents our widget.
        $widget = $vars['entity'];

        // Give the user a plain text box to input a message
        echo elgg_view('input/text', array(
            'name' => 'params[message]',
            'value' => $widget->message,
            'class' => 'hello-input-text',
        ));
    ?>
</div>
```

Notice the relationship between the values passed to the "name" and the "value" fields of input/text. The name of the input text box is `params[message]` because Elgg will automatically handle widget variables put in the array `params`. The actual php variable name will be `message`. If we wanted to use the field `greeting` instead of `message` we would pass the values `params[greeting]` and `$widget->greeting` respectively.

The reason we set the "value" option of the array is so that the edit view remembers what the user typed in the previous time he changed the value of his message text.

Now to display the user's message we need to modify content.php to use this *message* variable. Edit `/views/default/widgets/helloworld/content.php` and change it to:

```
<?php

$widget = $vars['entity'];

// Always use the corresponding output/* view for security!
echo elgg_view('output/text', array('value' => $widget->message));
```

You should now be able to enter a message in the text box and see it appear in the widget.

CAPÍTULO 5

Documentos de diseño

Aprende y comprende el funcionamiento interno de Elgg y los principios que lo motivan.

5.1 Actions

Actions are the primary way users interact with an Elgg site.

5.1.1 Resumen

An action in Elgg is the code that runs to make changes to the database when a user does something. For example, logging in, posting a comment, and making a blog post are actions. The action script processes input, makes the appropriate modifications to the database, and provides feedback to the user about the action.

5.1.2 Action Handler

Actions are registered during the boot process by calling `elgg_register_action()`. All actions URLs start with `action/` and are served by Elgg's front end controller through the action service. This approach is different from traditional PHP applications that send information to a specific file. The action service performs *CSRF security checks*, and calls the registered action script file, then optionally forwards the user to a new page. By using the action service instead of a single script file, Elgg automatically provides increased security and extensibility.

In Elgg 1.8 and before, actions were handled by an action handler script in `` `engine/handlers/action_handler.php ``. This required specific rewrite rules for URLs beginning with `/action/`.

See *Formularios y acciones* for details on how to register and construct an action. To look at the core actions, check out the directory /actions.

5.2 Base de datos

Una discusión a conciencia del diseño y la motivación del modelo de datos de Elgg.

Contents

5.2.1 Resumen

In Elgg, everything runs on a unified data model based on atomic units of data called entities.

Plugins are discouraged from interacting directly with the database, which creates a more stable system and a better user experience becase content created by different plugins can be mixed together in consistent ways. With this approach, plugins are faster to develop, and are at the same time much more powerful.

Every entity in the system inherits the `ElggEntity` class. This class controls access permissions, ownership

Existen dos maneras de extender las entidades con información adicional:

`Metadata`: This is information describing the entity, usually added by the author of the entity when the entity is created. For example, tags, an ISBN number, a file location, or source language is metadata.

`Annotations`: This is information about the entity, usually added by a third party after the entity is created. For example, ratings, likes, and votes are annotations. (Comments were before 1.9.)

5.2.2 Datamodel

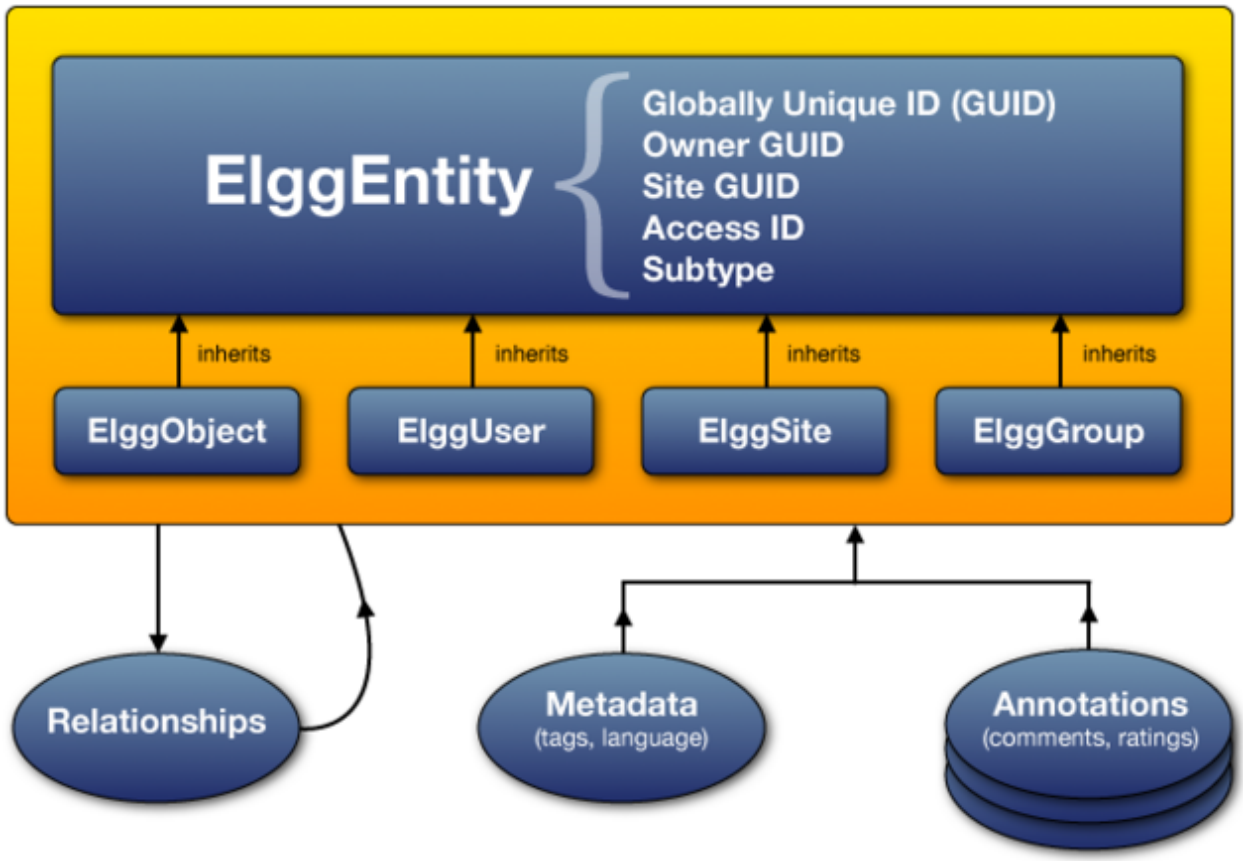

Figura 1: The Elgg data model diagram

5.2.3 Entidades

`ElggEntity` is the base class for the Elgg data model and supports a common set of properties and methods.

- Un identificador numérico único (véase *Identificadores únicos*).
- Permisos de acceso. Cuando un complemento solicita datos, no puede conseguir acceso a datos para los que el usuario actual no tiene permisos.
- An arbitrary subtype (more below).
- Un propietario.
- El sitio al que pertenece la entidad.
- A container, used to associate content with a group or a user.

Types

Actual entities will be instances of four different subclasses, each having a distinct **type** property and their own additional properties and methods.

Type	PHP class	Representa
object	`ElggObject`	Most user-created content, like blog posts, uploads, and bookmarks.
group	`ElggGroup`	An organized group of users with its own profile page
user	`ElggUser`	A system user
site	`ElggSite`	The site served by the Elgg installation

Each has its own extended API. E.g. objects have a `title` and `description`, users have a `username` and a way to set their password, and so on.

Subtypes

Each entity also has a custom string **subtype**, which plugins use to further specialize the entity. Elgg makes it easy to query specific subtypes as well as assign them special behaviors and views.

Subtypes are most commonly given to instances of ElggObject to denote the kind of content created. E.g. the blog plugin creates objects with subtype `"blog"`.

For historic reasons, the subtype API is a bit complex, but boils down to: write to `->subtype` before saving, otherwise always read `getSubtype()`. Below are more details.

Subtype Gotchas

- Before an entity's `save()` method is called, the subtype can be set by writing a string to the `subtype` property.
- *Subtype cannot be changed after saving.*
- After saving, you must always use `getSubtype()` to read it.
- If no subtype was set, `""` is returned, however some parts of the Elgg API (like Views) may map this value to the string `"default"`. E.g. a group with `getSubtype() === ""` will be rendered using the view `"group/default"`.
- Read carefully the documentation for `elgg_get_entities()` before trying to match subtypes; this
- API is a bit of a minefield. E.g. you cannot use `""` to fetch entities with the default subtype.

Identificadores únicos

Un identificador único (GUID) es un número entero que identifica de manera inequívoca cada entidad de una instalación de Elgg. Se asigna de manera automática cuando la entidad se guarda por primera vez, y no se puede volver a cambiar nunca.

Algunas funciones de la API de Elgg funcionan con identificadores en vez de con instancias de `ElggEntity`.

5.2.4 ElggObject

El tipo de entidad `ElggObject` representa contenido arbitrario dentro de la instalación de Elgg. Puede representar, por ejemplo, artículos de blog, ficheros, etc.

Además de las propiedades estándar de `ElggEntity`, `ElggObject` ofrece las siguientes propiedades:

- `title`: El título el objeto en texto escapado para HTML.
- `description` Una descripción del objeto en HTML.

La mayor parte del resto de los datos sobre el objeto se almacenan mediante metadatos.

5.2.5 ElggUser

El tipo de entidad `ElggUser` representa a un usuario de la instalación de Elgg. Estos usuarios se marcan como desactivados hasta que se activan sus cuentas, a menos que sean creados desde el panel de administración.

Además de las propiedades estándar de `ElggEntity`, `ElggUser` ofrece las siguientes propiedades:

- `name`: El nombre del usuario en texto plano. Por ejemplo: «Antonio García Fernández».
- `username`: Nombre con el que el usuario accede al sistema. Por ejemplo, «antoniogf».
- `password`: Una suma (hash) de la contraseña.
- `email`: La dirección de correo electrónico.
- `language`: El código del idioma predeterminado.
- `code`: El código de la sesión, que a partir de la versión 1.9 está en una tabla separada.
- `last_action`: La fecha en formato UNIX de la última vez que el usuario cargó una página.
- `prev_last_action`: El valor anterior de `last_action`.
- `last_login`: Fecha en formato UNIX de la última vez que el usuario accedió al sistema.
- `prev_last_login`: El valor anterior de `last_login`.

5.2.6 ElggSite

El tipo de entidad `ElggSite` representa sitios dentro de la instalación de Elgg. En la mayoría de las instalaciones sólo se dispone de un único sitio.

Además de las propiedades estándar de `ElggEntity`, `ElggSite` ofrece las siguientes propiedades:

- `name`: El nombre del sitio.
- `description`: Una descripción del sitio.
- `url`: La dirección URL del sitio.

5.2.7 ElggGroup

El tipo de entidad `ElggGroup` representa una asociación de usuarios de Elgg. Los usuarios pueden unirse a grupos, marcharse de ellos, o publicar contenido en ellos.

Además de las propiedades estándar de `ElggEntity`, `ElggGroup` ofrece las siguientes propiedades:

- `name`: El nombre del grupo en texto escapado para HTML.
- `description` Una descripción del grupo en HTML.

`ElggGroup` cuenta con métodos adicionales para gestionar su contenido y usuarios.

El complemento de grupos

Elgg incluye un complemento llamado «Grupos» que no debe confundirse con el tipo de entidad `ElggGroup`. El complemento provee la interfaz predeterminada con la que los usuarios del sitio pueden interactuar con los grupos. Cada grupo recibe un foro de discusiones y una página de perfil que pone a los usuarios en contacto con el contenido del grupo.

Usted puede cambiar la experiencia del usuario mediante los medios tradicionales de extender los complementos, o substituyendo el complemento de «Grupos» por su propio complemento.

Dado que `ElggGroup` puede tener un subtipo al igual que el resto de entidades, puede tener múltiples tipos de grupos en un mismo sitio.

Desarrollar un complemento compatible con grupos

Los desarrolladores de complementos no tienen por qué preocuparse mucho por que sus complementos sean compatibles con los grupos o sus funcionalidades, pero aquí van algunas claves a tener en cuenta:

Añadir contenido

Incluyendo el identificador del grupo como `container_guid` mediante un campo de entrada oculto, puede usar un mismo formulario para añadir contenido tanto a un usuario como a un grupo.

Use can_write_to_container para determinar si el usuario actual tiene o no permisos para añadir contenido a un grupo.

Tenga en cuenta que entonces tendrá que pasarle el identificador del contenedor o el nombre del usuario a la página responsable de publicar el contenido y el valor que lo acompañe, de forma que se pueda almacenar en su formulario como un campo de entrada oculto, para pasar fácilmente sus acciones. Dentro de una acción de crear, necesitará obtener este campo de entrada y guardarlo como una propiedad del nuevo elemento cuyo valor predeterminado es el contenedor del usuario actual:

```
$user = elgg_get_logged_in_user_entity();
$container_guid = (int)get_input('container_guid');
if ($container_guid) {
    if (!can_write_to_container($user->guid, $container_guid)) {
        // register error and forward
    }
} else {
    $container_guid = elgg_get_logged_in_user_guid();
}

$object = new ElggObject;
$object->container_guid = $container_guid;
```

(continué en la próxima página)

(proviene de la página anterior)

```
...

$container = get_entity($container_guid);
forward($container->getURL());
```

Nombres de usuario y propiedad de las páginas

Los grupos tienen un nombre de usuario simulado con la forma *group:identificador*, un valor al que se puede acceder mediante `$grupo->username`. Si pasa este nombre de usuario a una página en la línea de la URL como parte de la variable `username` (es decir, `/su_página?username=group:nnn`), Elgg registrará automáticamente la página como propiedad del grupo (salvo que se indique un propietario distinto de manera explícita).

Alternando entre usuarios y grupos

Lo cierto es que `ElggGroup` simula la mayor parte de los métodos de `ElggUser`. Puede obtener el icono del grupo, el nombre u otros datos con los mismos métodos, y si pide los contactos de un grupo, obtendrá sus miembros. Esto ha sido diseñado de manera específica para que usted pueda alternar entre usuarios y grupos de manera sencilla desde código.

Opciones de menú

Esta sección está obsoleta desde Elgg 1.8

Para los grupos predeterminados, la pieza final del rompecabezas es añadir un enlace a la funcionalidad que usted desarrolla desde el perfil del grupo. A continuación se mostrará cómo utilizando el complemento de ficheros como ejemplo.

Esto supone crear una vista dentro de su complemento —en este caso «fichero/menú»— que extenderá el menú del grupo. «Fichero/menú» consiste en un enlace entre etiquetas de párrafo que apunta al repositorio de ficheros del `page\_owner()`:

```
<p>
  <a href="<?php echo $vars['url']; ?>pg/file/<?php echo page_owner_entity()->
↪username; ?>">
    <?php echo elgg_echo("file"); ?>
  </a>
</p>
```

A continuación puede extender la vista del menú del grupo con la siguiente, dentro de la función de entrada de su complemento (en este caso `file\_init`):

```
extend_view('groups/menu/links', 'file/menu');
```

5.2.8 Propiedad

Las entidades tienen una propiedad, `owner_guid`, cuyo valor es el identificador único del propietario de la entidad. Habitualmente el identificador corresponde a un usuario, aunque tanto el propio sitio como sus usuarios no suelen tener propietario (el valor de la propiedad es `0`).

La propiedad de una entidad determina, en parte, si un usuario puede o no acceder a la entidad o modificarla.

5.2.9 Contenedores

Para poder buscar contenido fácilmente por usuario o por grupo, el contenido se define generalmente como «contenido» por el usuario que lo publicó o el grupo en el que se publicó. Eso significa que a la propiedad `container_guid` de los nuevos objetos se le dará como valor el identificador único de la instancia de `ElggUser` actual o de la instancia de `ElggGroup` de destino.

Por ejemplo, tres artículos de blog podrían pertenecer a autores distintos, pero estar todos contenidos por el grupo en el que se publicaron.

Nota: Esto no es siempre cierto. Las entidades de comentarios son propiedad del que comentan, y en algunos complementos de terceros el contenedor podría usarse para modelar una relación de padre-hijo entre entidades (por ejemplo, un objeto «carpeta» que contiene un objeto «fichero»).

5.2.10 Anotaciones

Las anotaciones son fragmentos de datos anexados a una entidad y que permiten a los usuarios indicar puntuaciones o añadir con ellas otro tipo de información relevante a la entidad a la que se añaden. Un complemento de encuestas podría, por ejemplo, registrar los votos como anotaciones. Antes de Elgg 1.9, los comentarios y las respuestas a discusiones de grupo se almacenaban como anotaciones.

Las anotaciones se almacenan como instancias de la clase `ElggAnnotation`.

Cada anotación cuenta con:

- Un tipo de anotación interno (como *comment*, «comentario»).
- Un valor (que puede ser una cadena de texto o un número entero).
- Un permiso de acceso distinto del de la entidad a la que está anexado.
- Un propietario.

Like metadata, values are stored as strings unless the value given is a PHP integer (`is_int($value)` is true), or unless the `$vartype` is manually specified as `integer`.

Añadir una anotación

La forma más sencilla de añadir una anotación a una entidad es usar el método `annotate` de dicha entidad. La declaración de dicho método es la siguiente:

```
public function annotate(
    $name,           // The name of the annotation type (eg 'comment')
    $value,          // The value of the annotation
    $access_id = 0,  // The access level of the annotation
    $owner_id = 0,   // The annotation owner, defaults to current user
    $vartype = ""    // 'text' or 'integer'
)
```

Por ejemplo, para dejar una puntuación en una entidad, puede llamar a:

```
$entity->annotate('rating', $rating_value, $entity->access_id);
```

Leer anotaciones

Para obtener las anotaciones de un objeto, puedes llamar al método siguiente:

```
$annotations = $entity->getAnnotations(
    $name,    // The type of annotation
    $limit,   // The number to return
    $offset,  // Any indexing offset
    $order,   // 'asc' or 'desc' (default 'asc')
);
```

Por si el tipo de tu anotación lidia en gran medida con números enteros, tiene a su disposición una serie de funciones matemáticas que pueden resultarle útiles:

```
$averagevalue = $entity->getAnnotationsAvg($name);  // Get the average value
$total = $entity->getAnnotationsSum($name);         // Get the total value
$minvalue = $entity->getAnnotationsMin($name);      // Get the minimum value
$maxvalue = $entity->getAnnotationsMax($name);      // Get the maximum value
```

Funciones de asistencia útiles

Comentarios

Si quiere ofrecer la funcionalidad de dejar comentarios en los objetos de su complemento, la siguiente función se lo facilitará, formulario y acciones:

```
function elgg_view_comments(ElggEntity $entity)
```

5.2.11 Metadatos

En Elgg, los metadatos le permiten almacenar alguna información adicional en una `entidad` más allá de los campos predeterminados que ofrece la entidad. Por ejemplo, las instancias de `ElggObject` sólo ofrecen los campos de entidad básicos y campos para un título y una descripción, pero puede que quieras incluir etiquetas o un número ISBN. De manera similar, puede que quieras que los usuarios tengan la posibilidad de guardar una fecha de cumpleaños.

Internamente, los metadatos se almacenan como una instancia de `ElggMetadata`, pero usted no tiene que preocuparse de eso en la práctica (pero si le interesa, échele una ojeada la la documentación de referencia de la clase). Lo único que necesita saber es que:

- Metadata has an owner and access ID (see note below), both of which may be different to the owner of the entity it's attached to
- Usted podría tener varios elementos de cata tipo de metadatos anexados a una única entidad.
- Like annotations, values are stored as strings unless the value given is a PHP integer (`is_int($value)` is true), or unless the `$value_type` is manually specified as `integer` (see below).

Nota: Metadata's `access_id` value will be ignored in Elgg 3.0 and all metadata values will be available in all contexts.

El caso más simple

Añadir metadatos

Para añadir un metadato a una entidad, llama a:

```
$entity->metadata_name = $metadata_value;
```

Por ejemplo, para añadir una fecha de cumpleaños a un usuario:

```
$user->dob = $dob_timestamp;
```

O para añadir un par de etiquetas a un objeto:

```
$object->tags = array('tag one', 'tag two', 'tag three');
```

Al añadir metadatos mediante el procedimiento anterior:

- El usuario actual se convierte en el propietario de los metadatos.
- Access permissions are inherited from the entity (see note below)
- Al reasignar un metadato, este substituye cualquier valor previo de ese mismo metadato.

Esto funciona bien para la mayoría de los casos. Tenga cuidado de anotar qué atributos son metadatos y cuales son parte integral del tipo de entidad con el que está trabajando. No tiene por qué guardar una entidad después de añadirle o actualizar sus metadatos. Pero si lo que ha cambiado son sus atributos integrales, debe guardar la entidad para preservar sus cambios. Por ejemplo, si ha cambiado el identificador de acceso de un objeto de tipo `ElggObject`, tiene que guardar el objeto, o de lo contrario el cambio no se hará efectivo en la base de datos.

Nota: Metadata's `access_id` value will be ignored in Elgg 3.0 and all metadata values will be available in all contexts.

Leer metadatos

Para obtener un metadato, haga lo mismo que haría con una propiedad de la entidad:

```
$tags_value = $object->tags;
```

Nota: El método anterior devolverá el valor absoluto del metadato. Para obtener el metadato como una instancia de `ElggMetadata`, use los métodos descritos en la sección *Un mayor control* más adelante.

Si almacena varios valores en este metadato (como en el ejemplo de las etiquetas), obtendrá un vector (array) con todos esos valores. Si sólo almacenó un valor, obtendrá una cadena de texto o un número entero. Si almacenó un vector con un único valor, el metadato devolverá una cadena de texto. Por ejemplo:

```
$object->tags = array('tag');
$tags = $object->tags;
// $tags will be the string "tag", NOT array('tag')
```

Para obtener siempre un vector, no tiene más que invocar el método como tal:

```
$tags = (array)$object->tags;
```

Un mayor control

Añadir metadatos

Si necesita más control para, por ejemplo, asignar un identificador de acceso distinto del identificador predeterminado, puede usar la función `create_metadata`, que está declarada así:

```
function create_metadata(
    $entity_guid,           // The GUID of the parent entity
    $name,                  // The name of the metadata (eg 'tags')
    $value,                 // The metadata value
    $value_type,            // Currently either 'text' or 'integer'
    $owner_guid,            // The owner of the metadata
    $access_id = 0,         // The access restriction
    $allow_multiple = false // Do we have more than one value?
    )
```

Para valores individuales, puede escribir el metadato de la siguiente manera (siguiendo con el ejemplo de la fecha de cumpleaños anexada a un usuario):

```
create_metadata($user_guid, 'dob', $dob_timestamp, 'integer', $_SESSION['guid'],
↪$access_id);
```

Nota: `$access_id` will be ignored in Elgg 3.0 and all metadata values will be available in all contexts. Always set it to `ACCESS_PUBLIC` for compatibility with Elgg 3.0.

Para varios valores, tendrá que iterar por ellos y llamar a `create_metadata` una vez por cada valor. El siguiente código ha sido extraído de Elgg, concretamente de la acción de guardar de los perfiles:

```
$i = 0;
foreach ($value as $interval) {
    $i++;
    $multiple = ($i != 1);
    create_metadata($user->guid, $shortname, $interval, 'text', $user->guid, $access_
↪id, $multiple);
}
```

Nota: El parámetro *allow_múltiple* (permitir varios) se pone a *false* en la primera iteración y a *true* (verdadero) en las siguientes.

Leer metadatos

`elgg_get_metadata` es la mejor función para obtener metadatos como instancias de `ElggMetadata`:

Por ejemplo, para obtener la fecha de cumpleaños (*DOB*) de un usuario:

```
elgg_get_metadata(array(
    'metadata_name' => 'dob',
    'metadata_owner_guid' => $user_guid,
));
```

O para obtener todos los objetos de metadatos:

```
elgg_get_metadata(array(
    'metadata_owner_guid' => $user_guid,
    'limit' => 0,
));
```

Errores habituales

"Anexar" metadatos

No puede "anexar" valores a vectores de metadatos como si fuesen simples vectores de PHP. Por ejemplo, el siguiente código no hace lo que parece que debería hacer:

```
$object->tags[] = "tag four";
```

Intentar almacenar mapas de sumas («hashmaps»)

Elgg no permite almacenar mapas ordenados (parejas con nombre y valor) en metadatos. Por ejemplo, el siguiente código no funcionaría como uno cabría esperar en un principio:

```
// Won't work!! Only the array values are stored
$object->tags = array('one' => 'a', 'two' => 'b', 'three' => 'c');
```

En vez de eso, puede hacer lo siguiente para almacenar la información:

```
$object->one = 'a';
$object->two = 'b';
$object->three = 'c';
```

Almacenar identificadores únicos en metadatos

Aunque existen algunos casos en los que tiene sentido almacenar identificadores únicos en metadatos, las *relaciones* son una construcción mucho mejor para relacionar unas entidades con otras.

5.2.12 Relaciones

Las relaciones permiten asociar entidades las unas con las otras. Por ejemplo: un artista que tiene fans, un usuario que es miembro de una organización, etc.

La clase `ElggRelationship` modela una relación con dirección entre dos entidades, dando lugar a la expresión:

> «**{sujeto}** es un **{substantivo}** de **{objeto}**.»

Nombre en el API	Modela	Representa
`guid_one`	Sujeto	La entidad que se está relacionando.
`relationship`	Substantivo	El tipo de la relación.
`guid_two`	Objeto	La entidad con la que se relaciona el sujeto.

Alternativamente, el tipo de relación puede ser un verbo, dando lugar a la expresión:

> «**{sujeto} {verbo} {objeto}**.»
>
> Por ejemplo, el usuario A «está de acuerdo con» el comentario B.

Toda relación tiene una dirección. Imagínese un arquero disparando una flecha al objeto. La flecha se mueve en una dirección, relacionando el sujeto (el arquero) con el objeto.

Las relaciones no implicar reciprocidad. Que **A** siga a **B** no significa que **B** siga a **A**.

Las relaciones_ carecen de control de acceso. Nunca se les ocultan a las vistas, y pueden editarse mediante código independientemente del nivel de privilegios, con la pega de que las *entidades* de la relación podrían sí ser invisibles por culpa del control de acceso.

Trabajar con relaciones

Crear una relación

Por ejemplo, para establecer que «**$user** es un **fan** de **$artist**», donde $user (usuario) es el sujeto y $artist (artista) el objeto:

```
// option 1
$success = add_entity_relationship($user->guid, 'fan', $artist->guid);

// option 2
$success = $user->addRelationship($artist->guid, 'fan');
```

Esto desencadena el evento [create, relationship], pasando a éste el objeto de tipo `ElggRelationship` creado. Si un manejador devuelve `false`, la relación no se creará y `$success` (que indica si todo fue correctamente) pasará a ser `false`.

Verificar una relación

Por ejemplo, para verificar que «**$user** es **fan** de **$artist**»:

```
if (check_entity_relationship($user->guid, 'fan', $artist->guid)) {
    // relationship exists
}
```

Tenga en cuenta que, si la relación existe, `check_entity_relationship()` devuelve una instancia de `ElggRelationship`:

```
$relationship = check_entity_relationship($user->guid, 'fan', $artist->guid);
if ($relationship) {
    // use $relationship->id or $relationship->time_created
}
```

Eliminar una relación

Por ejemplo, para poder asegurarse de que «**$user** ya no es **fan** de **$artist**»:

```
$was_removed = remove_entity_relationship($user->guid, 'fan', $artist->guid);
```

Esto desencadena el evento [delete, relationship], pasando a éste el objeto de tipo `ElggRelationship` asociado. Si un manejador devuelve `false`, la relación seguirá existiendo y `$was_removed` («se eliminó») pasará a ser `false`.

Otras funciones útiles:

- `delete_relationship()` : eliminar la relación con el identificador indicado.
- `remove_entity_relationships()` : eliminar aquellas entidades que estén relacionadas con una entidad determinada. Nota: en versiones de Elgg anteriores a la 1.9 esto no desencadenaba los eventos de eliminación.

Encontrar relaciones y entidades relacionadas

A continuación se presentan algunas funciones para obtener objetos de relaciones y entidades relacionadas:

- `get_entity_relationships()`: obtener relaciones que tengan la entidad indicada como sujeto o objeto.

- `get_relationship()`: obtener el objeto de relación con el identificador indicado.
- `elgg_get_entities_from_relationship()`: obtener entidades en relaciones de varias formas.

E.g. retrieving users who joined your site in January 2014.

```
$entities = elgg_get_entities_from_relationship(array(
    'relationship' => 'member_of_site',
    'relationship_guid' => elgg_get_site_entity()->guid,
    'inverse_relationship' => true,

    'relationship_created_time_lower' => 1388534400, // January 1st 2014
    'relationship_created_time_upper' => 1391212800, // February 1st 2014
));
```

5.2.13 Control de acceso

Los controles de acceso granular son uno de los principio de diseño fundamentales de Elgg, y una funcionalidad que ha estado en el centro de el sistema a lo largo de su desarrollo. La idea es sencilla: un usuario debería tener todo el control sobre quién puede ver un dato que ha creado dicho usuario.

Controles de acceso en el modelo de datos

Para conseguir esto, toda entidad, anotación y metadato contiene una propiedad `access_id` (identificador de acceso) que se corresponde con uno de los controles de acceso predefinidos o una entrada en la tabla `access_collections` de la base de datos.

Controles de acceso predefinidos

- `ACCESS_PRIVATE` (value: 0) Private.
- `ACCESS_LOGGED_IN` (value: 1) Logged in users.
- `ACCESS_PUBLIC` (value: 2) Public data.
- `ACCESS_FRIENDS` (value: -2) Owner and his/her friends.

Controles de acceso definidos por los usuarios

Usted puede definir grupos de acceso adicionales y asignárselos a entidades, anotaciones o metadatos. Existen para ayudarle a ello una serie de funciones; para más información, vea la documentación de referencia de la biblioteca de acceso.

Cómo el acceso afecta a la obtención de datos

Todas las funciones de obtención de datos que hay sobre la capa de la base de datos —como `get_entities` y similares— sólo devuelven elementos a los que el usuario actual tiene acceso de lectura. No es posible obtener elementos a los que el usuario actual no tiene acceso. Esto dificulta el crear una brecha de seguridad en la obtención de datos.

Acceso de escritura

Las siguientes reglar rigen el acceso de escritura:

- El propietario de una entidad siempre puede editarla.
- El propietario de un contenedor puede editar todo lo que hay en el contenedor. Nota: Ello no significa que el propietario de un grupo pueda editar cualquier cosa dentro del grupo).
- Los administradores pueden editarlo todo.

You can override this behaviour using a *plugin hook* called `permissions_check`, which passes the entity in question to any function that has announced it wants to be referenced. Returning `true` will allow write access; returning `false` will deny it. See *the plugin hook reference for permissions_check* for more details.

Ver también:

Documentación de referencia de la biblioteca de acceso

5.2.14 Esquema

La base de datos contiene una serie de tablas primarias y secundarias. Su esquema de tabla está almacenado en `/engine/schema/mysql.sql`.

Cada tabla está prefijada con «prefix_», que la infraestructura de Elgg substituye por el prefijo seleccionado durante la instalación.

Tablas principales

Esta es una descripción de las tablas principales. No se olvide de que en toda instalación de Elgg estas tablas tienen un prefijo, que suele ser «elgg_».

Tabla: entidades

Esta es la tabla principal de entidades, y contiene a los usuarios de Elgg, los sitios, los objetos y los grupos. Cuando instala Elgg por primera vez, la tabla se rellena automáticamente con su primer sitio.

Contiene los siguientes campos:

- **guid:** Un contador que aumenta automáticamente y produce un identificador único para la entidad dentro del sistema.
- **type:** El tipo de la entidad (`object`, `user`, `group` o `site`).
- **subtype** A reference to the *entity_subtypes* table, or `0` for the default subtype.
- **owner_guid:** El identificador único del propietario de la entidad.
- **site_guid:** El sitio al que pertenece la entidad.
- **container_guid:** El identificador único de la entidad que contiene a esta, que puede ser un usuario o un grupo.
- **access_id:** Los controles de acceso de la entidad.
- **time_created:** La fecha y hora de creación de la entidad en formato Unix.
- **time_updated:** La fecha y hora de la última actualización de la entidad en formato Unix.
- **enabled:** Si su valor es «yes» se puede acceder a la entidad, si es «no» la entidad está desactivada (Elgg la trata como si hubiese sido eliminada pero sin que se haya eliminado realmente de la base de datos).

Tabla: entity_subtypes

Esta tabla contiene información sobre los subtipos de las entidades:

- **id:** Un contador.
- **type:** El tipo de la entidad (`object`, `user`, `group` o `site`)..
- **subtype:** El nombre del subtipo en forma de cadena de texto.
- **class:** Nombre de clase opcional si el subtipo está enlazado con una clase.

Tabla: metadata

Esta tabla contiene *metadatos*, información adicional anexada a una entidad.

- **id:** Un contador.
- **entity_guid:** Identificador de la entidad a la que está anexado el metadato.
- **name_id:** Un enlace a la tabla «metastrings» que define el nombre.
- **value_id:** Un enlace a la tabla «metastrings» que define el valor.
- **value_type:** La clase del valor, o bien «text» (texto) o bien «integer» (número entero).
- **owner_guid:** El identificador único del propietario, es decir, de quien definió el metadato.
- **access_id:** Los controles de acceso del metadato.
- **time_created:** La fecha y hora de creación del metadato en formato Unix.
- **enabled:** Si su valor es «yes» se puede acceder al elemento, si es «no» el elemento ha sido eliminado.

Tabla: annotations

Esta tabla contiene *anotaciones*, que no son lo mismo que *metadatos*.

- **id:** Un contador.
- **entity_guid:** Identificador de la entidad a la que está anexado el metadato.
- **name_id:** Un enlace a la tabla «metastrings» que define el tipo de anotación.
- **value_id:** Un enlace a la tabla «metastrings» que define el valor.
- **value_type:** La clase del valor, o bien «text» (texto) o bien «integer» (número entero).
- **owner_guid:** El identificador único del propietario, es decir, de quien definió el metadato.
- **access_id:** Los controles de acceso del metadato.
- **time_created:** La fecha y hora de creación del metadato en formato Unix.
- **enabled:** Si su valor es «yes» se puede acceder al elemento, si es «no» el elemento ha sido eliminado.

Tabla: relationships

Esta tabla define *relaciones*, las cuales enlazan unas entidades con otras.

- **guid_one:** El identificador de la entidad sujeto.
- **relationship:** El tipo de la relación.

- **guid_two:** El identificador de la entidad objeto.

Tabla: objects_entity

Información adicional relacionada específicamente con objetos. Estos están divididos para reducir la carga de la tabla de metadatos y hacer una diferencia obvia entre los atributos y los metadatos.

Tabla: sites_entity

Información adicional relacionada específicamente con sitios. Estos están divididos para reducir la carga de la tabla de metadatos y hacer una diferencia obvia entre los atributos y los metadatos.

Tabla: users_entity

Información adicional relacionada específicamente con usuarios. Estos están divididos para reducir la carga de la tabla de metadatos y hacer una diferencia obvia entre los atributos y los metadatos.

Tabla: groups_entity

Información adicional relacionada específicamente con grupos. Estos están divididos para reducir la carga de la tabla de metadatos y hacer una diferencia obvia entre los atributos y los metadatos.

Tabla: metastrings

Las metastrigns (metacadenas) contiene los textos reales de los metadatos, y que están enlazados con las tablas de metadatos y de anotaciones.

Esto permite evitar duplicar cadenas, ahorrando espacio y haciendo más eficientes las búsquedas en la base de datos.

5.3 Eventos y ganchos de complementos

Contents

5.3.1 Resumen

Elgg cuenta con un sistema de eventos que puedes usar para substituir o extender su funcionalidad predeterminada.

Los complementos influencian el sistema mediante la creación de manejadores (llamadas de retorno como funciones o métodos) y registrándolos para manejar dos tipos de eventos: *eventos de Elgg* y *ganchos de complementos*.

When an event is triggered, a set of handlers is executed in order of priority. Each handler is passed arguments and has a chance to influence the process. After execution, the «trigger» function returns a value based on the behavior of the handlers.

Diferencias entre los eventos de Elgg y los ganchos de complementos

The main differences between *Elgg Events* and *Plugin Hooks* are:

1. La mayor parte de los eventos de Elgg se pueden cancelar. A menos que los eventos sean eventos «after» (que tienen lugar después de que suceda algo), los manejadores de los eventos pueden cancelar dichos eventos devolviendo `false`, y con ello evitarán que se llame al resto de manejadores.
2. Los ganchos de complementos no pueden cancelarse; todos los ganchos se llaman siempre.
3. Los ganchos de complementos pasan un valor arbitrario a través de los manegadores, dándole a cada uno de ellos la oportunidad de modificarlo.

5.3.2 Eventos de Elgg

Los eventos de Elgg se desencadenan cuando se crea, actualizar o elimina un objeto. También tienen lugar en puntos importantes del proceso de carga de la infraestructura Elgg. Por ejemplo, existen eventos que suceden al crear un artículo de blog o al acceder un usuario al sitio.

Unlike *Plugin Hooks*, *most Elgg events can be cancelled*, halting the execution of the handlers, and possibly cancelling an some action in the Elgg core.

Todos los eventos de Elgg tienen un nombre y un tipo de objeto (`system`, `user`, `object`, `relationship`, `annotation`, `group`) que describe el tipo del objeto que se pasa a los manejadores.

Eventos del antes y el después

Some events are split into «before» and «after». This avoids confusion around the state of the system while in flux. E.g. Is the user logged in during the [login, user] event?

Before Events have names ending in «:before» and are triggered before something happens. Like traditional events, handlers can cancel the event by returning `false`.

Los eventos del después, con nombres que terminan en «:after», ocurren después de que algo suceda. A diferencia de los eventos tradicionales, sus manejadores *no* pueden cancelar estos eventos; se llama siempre a todos los manejadores.

Cuando para un evento existen tanto la versión del antes como la del después, se recomienda a los desarrolladores que usen estas versiones del evento en vez de la principal —la que no es ni del antes ni del después—, si bien se seguirá permitiendo el uso de la versión original del evento para que Elgg siga siendo compatible con versiones anteriores de la infraestructura.

Manejadores de eventos de Elgg

Los manejadores de eventos de Elgg deberían tener el siguiente prototipo:

```
/**
 * @param string $event       The name of the event
 * @param string $object_type The type of $object (e.g. "user", "group")
 * @param mixed  $object      The object of the event
 *
 * @return bool if false, the handler is requesting to cancel the event
 */
function event_handler($event, $object_type, $object) {
    ...
}
```

If the handler returns `false`, the event is cancelled, preventing execution of the other handlers. All other return values are ignored.

Registrar un manejador para un evento de Elgg

Registre su manejador para un evento mediante `elgg_register_event_handler`:

```
elgg_register_event_handler($event, $object_type, $handler, $priority);
```

Parámetros:

- **$event:** El nombre del evento.
- **$object_type:** El tipo de objeto (por ejemplo: «user» o «object») en el que se lanza el evento, «all» si son todos los tipos.
- **$handler:** La llamada de retorno de la función manejadora.
- **$priority:** La prioridad. 0 es la mayor prioridad, 500 es el valor predeterminado.

Object here does not refer to an `ElggObject` but rather a string describing any object in the framework: system, user, object, relationship, annotation, group.

Ejemplo:

```
// Register the function myPlugin_handle_login() to handle the
// user login event with priority 400.
elgg_register_event_handler('login', 'user', 'myPlugin_handle_login', 400);
```

Advertencia: If you handle the «update» event on an object, avoid calling `save()` in your event handler. For one it's probably not necessary as the object is saved after the event completes, but also because `save()` calls another «update» event and makes `$object->getOriginalAttributes()` no longer available.

Desencadenar un evento de Elgg

Usted puede desencadenar un evento personalizado de Elgg mediante `elgg_trigger_event`:

```
if (elgg_trigger_event($event, $object_type, $object)) {
    // Proceed with doing something.
} else {
    // Event was cancelled. Roll back any progress made before the event.
}
```

For events with ambiguous states, like logging in a user, you should use *Before and After Events* by calling `elgg_trigger_before_event` or `elgg_trigger_after_event`. This makes it clear for the event handler what state to expect and which events can be cancelled.

```
// handlers for the user, login:before event know the user isn't logged in yet.
if (!elgg_trigger_before_event('login', 'user', $user)) {
        return false;
}

// handlers for the user, login:after event know the user is logged in.
elgg_trigger_after_event('login', 'user', $user);
```

Parámetros:

- **$event:** El nombre del evento.
- **$object_type:** El tipo de objeto (por ejemplo: «user» o «object»).
- **$object:** El objeto (como puede ser una instancia de `ElggUser` o de `ElggGroup`).

The function will return `false` if any of the selected handlers returned `false` and the event is stoppable, otherwise it will return `true`.

5.3.3 Ganchos de complementos

Plugin Hooks provide a way for plugins to collaboratively determine or alter a value. For example, to decide whether a user has permission to edit an entity or to add additional configuration options to a plugin.

A plugin hook has a value passed into the trigger function, and each handler has an opportunity to alter the value before it's passed to the next handler. After the last handler has completed, the final value is returned by the trigger.

Manejadores de ganchos de complementos

Los manejadores de ganchos de complementos deberían tener el siguiente prototipo:

```
/**
 * @param string $hook    The name of the plugin hook
 * @param string $type    The type of the plugin hook
 * @param mixed  $value   The current value of the plugin hook
 * @param mixed  $params  Data passed from the trigger
 *
 * @return mixed if not null, this will be the new value of the plugin hook
 */
function plugin_hook_handler($hook, $type, $value, $params) {
    ...
}
```

If the handler returns no value (or `null` explicitly), the plugin hook value is not altered. Otherwise the return value becomes the new value of the plugin hook. It will then be passed to the next handler as `$value`.

Registrar un manejador para un gancho de complemento

Para registrar un manejador para un gancho de complementos, use `elgg_register_plugin_hook_handler`:

```
elgg_register_plugin_hook_handler($hook, $type, $handler, $priority);
```

Parámetros:

- **$hook:** El nombre del gancho de complementos.
- **$type:** El tipo del gancho, o «all» si son todos los tipos.
- **$handler:** La llamada de retorno de la función manejadora.
- **$priority:** La prioridad. 0 es la mayor prioridad, 500 es el valor predeterminado.

El significado del **tipo** puede variar. Podría tratarse del tipo de una entidad de Elgg o de algo específico del nombre del gancho de complementos.

Ejemplo:

```
// Register the function myPlugin_hourly_job() to be called with priority 400.
elgg_register_plugin_hook_handler('cron', 'hourly', 'myPlugin_hourly_job', 400);
```

Desencadenar un gancho de componente

Usted puede desencadenar un gancho de componente personalizado mediante `elgg_trigger_plugin_hook`:

```
// filter $value through the handlers
$value = elgg_trigger_plugin_hook($hook, $type, $params, $value);
```

Parámetros:

- **$hook:** El nombre del gancho de complementos.
- **$type:** El tipo del gancho, o «all» si son todos los tipos.
- **$params:** Datos arbitrarios que se pasaron del desencadenante a los manejadores.
- **$value:** El valor inicial del gancho de complementos.

Advertencia: The *$params* and *$value* arguments are reversed between the plugin hook handlers and trigger functions!

Unregister Event/Hook Handlers

The functions `elgg_unregister_event_handler` and `elgg_unregister_plugin_hook_handler` can be used to remove handlers already registered by another plugin or Elgg core. The parameters are in the same order as the registration functions, except there's no priority parameter.

```
elgg_unregister_event_handler('login', 'user', 'myPlugin_handle_login');
```

Anonymous functions or invokable objects cannot be unregistered, but dynamic method callbacks can be unregistered by giving the static version of the callback:

```
$obj = new MyPlugin\Handlers();
elgg_register_plugin_hook_handler('foo', 'bar', [$obj, 'handleFoo']);

// ... elsewhere

elgg_unregister_plugin_hook_handler('foo', 'bar', 'MyPlugin\Handlers::handleFoo');
```

Even though the event handler references a dynamic method call, the code above will successfully remove the handler.

Handler Calling Order

Handlers are called first in order of priority, then registration order.

Nota: Before Elgg 2.0, registering with the `all` keywords caused handlers to be called later, even if they were registered with lower priorities.

5.4 Internacionalización

La versión 1.0 de Elgg marcó una diferencia con las anteriores en cuanto a internacionalización: Elgg 1.0 y versiones posteriores usan un vector de texto personalizado en vez de Gettext. Esto mejora la eficiencia del sistema y hace más fiable el sistema de traducción.

Por hacer: Más información, por favor.

5.5 AMD

5.5.1 Resumen

En Elgg hay dos sistemas de JavaScript: el sistema de la versión 1.8, obsoleto, y el nuevo sistema compatible con AMD y fue introducido en la versión 1.9.

A continuación se discuten los beneficios de usar AMD en Elgg.

5.5.2 ¿Por qué AMD?

We have been working hard to make Elgg's JavaScript more maintainable and useful. We made some strides in 1.8 with the introduction of the «`elgg`» JavaScript object and library, but have quickly realized the approach we were taking was not scalable.

The size of JS on the web is growing quickly, and JS in Elgg is growing too. We want Elgg to be able to offer a solution that makes JS development as productive and maintainable as possible going forward.

Los motivos para elegir AMD son muchos y están bien documentados. A continuación nos limitaremos a subrayar algunos de los más relevantes y más relacionados con Elgg.

1. Simplifica la gestión de dependencias.

Los módulos de AMD se cargan de manera asíncrona y se ejecutan en cuanto sus dependencias están disponibles, lo que elimina la necesidad de especificar la «prioridad» o la «localización» a la hora de registrar bibliotecas JavaScript en Elgg. Además, no tiene que preocuparse de cargar explícitamente las dependencias de un módulo desde PHP. El cargador de AMD (RequireJS en este caso) se encarga de todo ello por usted. También se pueden tener dependencias de texto con el complemento de texto de RequireJS, de forma que utilizar modelos del lado del cliente será pan comido.

2. AMD ya funciona en todos los navegadores.

Los desarrolladores de Elgg ya están escribiendo un montón de JavaScript. Y sabemos que usted quiere escribir más. No podemos aceptar esperarnos entre 5 y 10 años a que esté disponible en todos los navegadores una solución de módulos de JavaScript nativos para ponernos a organizar nuestro JavaScript para que sea fácil de mantener.

3. No hace falta ningún paso de construcción para desarrollar con AMD.

We like the edit-refresh cycle of web development. We wanted to make sure everyone developing in Elgg could continue experiencing that joy. Synchronous module formats like Closure or CommonJS just weren't an option for us. But even though AMD doesn't require a build step, *it is still very build-friendly*. Because of the `define()` wrapper, it's possible to concatenate multiple modules into a single file and ship them all at once in a production environment.[1]

AMD es un sistema de carga de módulos a prueba de balas y bien diseñado, pensado para la web actual. Estamos muy agradecidos por el esfuerzo que sus desarrolladores han puesto en él, y estamos emocionados ante la idea de ofrecerlo como la solución estándar para el desarrollo con JavaScript en Elgg empezando con la versión 1.9.

5.6 Seguridad

La forma en que Elgg aborda los diversos problemas de seguridad es la misma que la del resto de aplicaciones web.

Truco: Para informar de una posible vulnerabilidad en Elgg, envíe un correo electrónico a security@elgg.org.

Contents

[1] Esto aún no es compatible con el núcleo de Elgg, pero estamos investigando cómo solucionarlo, ya que no tener que dar tantas vueltas resulta fundamental para una buena primera impresión, sobre todo en dispositivos móviles.

5.6.1 Contraseñas

Validación de contraseñas

The only restriction that Elgg places on a password is that it must be at least 6 characters long by default, though this may be changed in `/elgg-config/settings.php`. Additional criteria can be added by a plugin by registering for the `registeruser:validate:password` plugin hook.

Suma de la contraseña

Passwords are never stored, only salted hashes produced with bcrypt. This is done via the standard `password_hash()` function. On older systems, the `password-compat` polyfill is used, but the algorithm is identical.

Elgg installations created before version 1.10 may have residual «legacy» password hashes created using salted MD5. These are migrated to bcrypt as users log in, and will be completely removed when a system is upgraded to Elgg 3.0. In the meantime we're happy to assist site owners to manually remove these legacy hashes, though it would force those users to reset their passwords.

Regulador de accesos

Elgg cuenta con un mecanismo de regulación de accesos que dificulta en gran medida los ataques de diccionario desde el exterior. Los usuarios sólo pueden hacer hasta 5 intentos de acceder al sistema durante un período de 5 minutos.

Restablecimiento de contraseñas

Si un usuario se olvida de su contraseña, se puede solicitar una nueva contraseña generada aleatoriamente. Tras la solicitud, se envía un mensaje de correo electrónico al usuario con una URL única. Cuando el usuario visita la URL, se le envía una nueva contraseña aleatoria por correo electrónico.

5.6.2 Sesiones

Elgg usa la gestión de sesiones de PHP con manejadores personalizados. Los datos de las sesiones se almacenan en la base de datos. La cookie de la sesión contiene el identificador de la sesión, que asocia el usuario con el navegador. Los metadatos del usuario se almacenan en la sesión, incluyendo su identificador, nombre de usuario y dirección de correo electrónico. El tiempo de vida de la sesión se controla desde la configuración de PHP en el servidor.

Fijación de la sesión

Elgg le protege contra la fijación de la sesión mediante la generación de un nuevo identificador de sesión cada vez que accede al sitio.

Robo de sesión

Advertencia: Esta sección resulta cuestionable.

Además de proteger contra ataques de fijación de sesión, Elgg cuenta con una comprobación adicional que intenta evitar el robo de sesión si el identificador de la sesión se ve comprometido. Elgg almacena una suma (hash) de la información del navegador web («user agent») y un secreto del sitio como huella dactilar de la sesión. El uso del secreto del sitio es un poco superfluo, pero la comprobación de la información del navegador podría evitar algunos intentos de robo de sesión.

Cookie de «recordarme»

Para permitir que los usuario permanezcan conectados durante más tiempo independientemente de que hayan cerrado el navegador web, Elgg utiliza una cookie (llamada «elggperm») que contiene lo que podría considerarse como un identificador de «super sesión». Este identificador se almacena en una tabla de cookies. Si existe y el código de la sesión de la cookie se corresponde con el código de la tabla de la cookie, el usuario correspondiente se identifica en el sitio de manera automática.

5.6.3 Autenticación alternativa

Nota: Esta sección no es muy precisa.

Para que un complemento pueda substituir el sistema predeterminado de autenticación de Elgg, dicho complemento debe substituir la acción predeterminada con la suya propia mediante `register_action()`. También tendría que registrar su propio manejador «pam» mediante `register_pam_handler()`.

Nota: La función `pam_authenticate()` que se usa para llamar a los diferentes módulos tienen un error relacionado con la variable de importancia.

5.6.4 HTTPS

Nota: Debe activar la compatibilidad con SSL en su servidor para que las siguientes técnicas funcionen.

Para que el formulario de acceso se envíe mediante HTTPS, active «login-over-ssl» desde el panel de administración de Elgg.

Para que todo el sitio utilice SSL, cambie la URL del sitio para que en vez de «http» empiece por «https».

5.6.5 XSS

En Elgg se utilizan filtros para dificultar ataques de secuencias de órdenes en sitios cruzados (XSS). EL propósito del filtrado es eliminar JavaScript y otros datos de entrada peligrosos de los usuarios.

El filtrado se realiza mediante la función `filter_tags()`. Esta función recibe una cadena de texto y devuelve la misma cadena pero filtrada. La función desencadena un gancho de complementos `validate, input`.

De manera predeterminada, Elgg se distribuye con el código del filtro «htmLawed» como complemento. Los desarrolladores pueden añadir más código de filtrado o substituir el existente mediante complementos.

A la función `filter_tags()` se la llama para todos los datos introducidos por usuarios siempre que dichos datos se obtuviesen mediante una llamada a la función `get_input()`. Si por lo que sea no se quiera realizar ese filtrado con algún dato introducido por un usuario, la función `get_input()` cuenta con un parámetro que permite desactivar dicho filtrado.

5.6.6 CSRF / XSRF

Elgg generates security tokens to prevent cross-site request forgery. These are embedded in all forms and state-modifying AJAX requests as long as the correct API is used. Read more in the *Formularios y acciones* developer guide.

5.6.7 Inyección SQL

La API de Elgg esteriliza todos los datos de entrada antes de ejecutar consultas contra la base de datos. Para más información, véase *Base de datos*.

5.6.8 Privacidad

Elgg usa un sistema de listas de control de acceso (ACL) para controlar qué usuarios tienen accedo a distintos contenidos. Para más información, véase *Base de datos*.

5.7 Loggable

Loggable is an interface inherited by any class that wants events relating to its member objects to be saved to the system log. `ElggEntity` and `ElggExtender` both inherit `Loggable`.

Loggable defines several class methods that are used in saving to the default system log, and can be used to define your own (as well as for other purposes):

- `getSystemLogID()` Return a unique identifier for the object for storage in the system log. This is likely to be the object's GUID
- `getClassName()` Return the class name of the object
- `getType()` Return the object type
- `getSubtype()` Get the object subtype
- `getObjectFromID($id)` For a given ID, return the object associated with it

5.7.1 Database details

The default system log is stored in the `system_log` *database table*. It contains the following fields:

- **id** - A unique numeric row ID
- **object_id** - The GUID of the entity being acted upon
- **object_class** - The class of the entity being acted upon (eg ElggObject)
- **object_type** - The type of the entity being acted upon (eg object)
- **object_subtype** - The subtype of the entity being acted upon (eg blog)
- **event** - The event being logged (eg create or update)
- **performed_by_guid** - The GUID of the acting entity (the user performing the action)
- **owner_guid** - The GUID of the user which owns the entity being acted upon
- **access_id** - The access restriction associated with this log entry
- **time_created** - The UNIX epoch timestamp of the time the event took place

CAPÍTULO 6

Contributor Guides

Participate in making Elgg even better.

Elgg is a community-driven project. It relies on the support of volunteers to succeed. Here are some ways you can help:

6.1 Translations

Translations multiply the impact that Elgg can have by making it accessible to a larger percentage of the world.

The community will always be indebted to those of you who work hard to provide high quality translations for Elgg's UI and docs.

6.1.1 Transifex

All translation for the Elgg project is organized through Transifex.

https://www.transifex.com/organization/elgg

Plugin authors are encouraged to coordinate translations via Transifex as well so the whole community can be unified and make it really easy for translators to contribute to any plugin in the Elgg ecosystem.

6.2 Reporting Issues

Report bugs and features requests to https://github.com/Elgg/Elgg/issues. See below for guidelines.

6.2.1 DISCLAIMERS

- **SECURITY ISSUES SHOULD BE REPORTED TO security @ elgg . org!** Please do not post any security issues on github!!

- Support requests belong on the community site. Tickets with support requests will be closed.
- We cannot make any guarantees as to when your ticket will be resolved.

6.2.2 Bug reports

Before submitting a bug report:

- Search for an existing ticket on the issue you're having. Add any extra info there.
- Verify the problem is reproducible
  - On the latest version of Elgg
  - With all third-party plugins disabled

Good bug report checklist:

- Expected behavior and actual behavior
- Clear steps to reproduce the problem
- The version of Elgg you're running
- Browsers affected by this problem

6.2.3 Feature requests

Before submitting a feature request:

- Check the community site for a plugin that has the features you need.
- Consider if you can *develop a plugin* that does what you need.
- Search through the closed tickets to see if someone else suggested the same feature, but got turned down. You'll need to be able to explain why your suggestion should be considered this time.

Good feature request checklist:

- Detailed explanation of the feature
- Real-life use-cases
- Proposed API

6.3 Writing Code

Understand Elgg's standards and processes to get your changes accepted as quickly as possible.

Contents

6.3.1 License agreement

By submitting a patch you are agreeing to license the code under a GPLv2 license and MIT license.

6.3.2 Pull requests

Pull requests (PRs) are the best way to get code contributed to Elgg core. The core development team uses them even for the most trivial changes.

For new features, submit a feature request or talk to us first and make sure the core team approves of your direction before spending lots of time on code.

Checklists

Use these markdown checklists for new PRs on github to ensure high-quality contributions and help everyone understand the status of open PRs.

Bugfix PRs:

```
- [ ] Commit messages are in the standard format
- [ ] Includes regression test
- [ ] Includes documentation update (if applicable)
- [ ] Is submitted against the correct branch
- [ ] Has LGTM from at least one core developer
```

Feature PRs:

```
- [ ] Commit messages are in the standard format
- [ ] Includes tests
- [ ] Includes documentation
- [ ] Is submitted against the correct branch
- [ ] Has LGTM from at least two core developers
```

Choosing a branch to submit to

The following table assumes the latest stable release is 2.1.

Type of change	Branch to submit against
Security fix	Don't! Email security@elgg.org for guidance.
Bug fix	1.12 (or 2.1 if the 1.12 fix is too complex)
Performance	2.x
Deprecation	2.x
Minor feature	2.x
Major feature	master
Has any breaking change	master

If you're not sure which branch to submit against, just ask!

The difference between minor and major feature is subjective and up to the core team.

Commit message format

We require a particular format to allow releasing more often, and with improved changelogs and source history. Just follow these steps:

1. Start with the `type` by selecting the *last category which applies* from this list:
   - **docs** - *only* docs are being updated
   - **chore** - this include refactoring, code style changes, adding missing tests, Travis stuff, etc.
   - **perf** - the primary purpose is to improve performance
   - **fix** - this fixes a bug
   - **deprecate** - the change deprecates any part of the API
   - **feature** - this adds a new user-facing or developer feature
   - **security** - the change affects a security issue in any way. *Please do not push this commit to any public repo.* Instead contact security@elgg.org.

   E.g. if your commit refactors to fix a bug, it's still a «fix». If that bug is security-related, however, the type must be «security» and you should email security@elgg.org before proceeding. When in doubt, make your best guess and a reviewer will provide guidance.

2. In parenthesis, add the `component`, a short string which describes the subsystem being changed.

   Some examples: `views`, `i18n`, `seo`, `a11y`, `cache`, `db`, `session`, `router`, `<plugin_name>`.

3. Add a colon, a space, and a brief `summary` of the changes, which will appear in the changelog.

   No line may exceed 100 characters in length, so keep your summary concise.

Good summary	Bad summary (problem)
page owners see their own owner blocks on pages	bug fix (vague)
bar view no longer dies if "foo" not set	updates views/default/bar.php so bar view no longer... (redundant info)
narrows river layout to fit iPhone	alters the river layout (vague)
elgg_foo() handles arrays for $bar	in elgg_foo() you can now pass an array for $bar and the function will... (move detail to description)
removes link color from comments header in river	fixes db so that... (redundant info)
requires non-empty title when saving pages	can save pages with no title (confusingly summarizes old behavior)

4. (recommended) Skip a line and add a `description` of the changes. Include the motivation for making them, any info about back or forward compatibility, and any rationale of why the change had to be done a certain way. Example:

   > We speed up the Remember Me table migration by using a single INSERT INTO ... SELECT query instead of row-by-row. This migration takes place during the upgrade to 1.9.

   Unless your change is trivial/obvious, a description is required.

5. If the commit resolves a GitHub issue, skip a line and add `Fixes #` followed by the issue number. E.g. `Fixes #1234`. You can include multiple issues by separating with commas.

   GitHub will auto-close the issue when the commit is merged. If you just want to reference an issue, use `Refs #` instead.

When done, your commit message will have the format:

```
type(component): summary

Optional body
Details about the solution.
Opportunity to call out as breaking change.

Closes/Fixes/Refs #123, #456, #789
```

Here is an example of a good commit message:

```
perf(upgrade): speeds up migrating remember me codes

We speed up the Remember Me table migration by using a single INSERT INTO ... SELECT␣
↪query instead of row-by-row.
This migration takes place during the upgrade to 1.9.

Fixes #6204
```

To validate commit messages locally, make sure `.scripts/validate_commit_msg.php` is executable, and make a copy or symlink to it in the directory `.git/hooks/commit-msg`.

```
chmod u+x .scripts/validate_commit_msg.php
ln -s .scripts/validate_commit_msg.php .git/hooks/commit-msg/validate_commit_msg.php
```

Rewriting commit messages

If your PR does not conform to the standard commit message format, we'll ask you to rewrite it.

To edit just the last commit:

1. Amend the commit: `git commit --amend` (git opens the message in a text editor).
2. Change the message and save/exit the editor.
3. Force push your branch: `git push -f your_remote your_branch` (your PR with be updated).
4. Rename the PR title to match

Otherwise you may need to perform an interactive rebase:

1. Rebase the last N commits: `git rebase -i HEAD~N` where N is a number. (Git will open the `git-rebase-todo` file for editing)
2. For the commits that need to change, change `pick` to `r` (for reword) and save/exit the editor.
3. Change the commit message(s), save/exit the editor (git will present a file for each commit that needs rewording).
4. `git push -f your_remote your_branch` to force push the branch (updating your PR).
5. Rename the PR title to match

6.3.3 Testing

Elgg has automated tests for both PHP and JavaScript functionality. All new contributions are required to come with appropriate tests.

General guidelines

Break tests up by the behaviors you want to test and use names that describe the behavior. E.g.:

- Not so good: One big method `testAdd()`.
- Better: Methods `testAddingZeroChangesNothing` and `testAddingNegativeNumberSubtracts`

Strive for *componentized designs* that allow testing in isolation, without large dependency graphs or DB access. Injecting dependencies is key here.

PHP Tests

PHPUnit

Located in `engine/tests/phpunit`, this is our preferred test suite. It uses no DB access, and has only superficial access to the entities API.

- We encourage you to create components that are testable in this suite if possible.
- Consider separating storage from your component so at least business logic can be tested here.
- Depend on the `Elgg\Filesystem\*` classes rather than using PHP filesystem functions.

SimpleTest

The rest of the files in `engine/tests` form our integration suite, for anything that needs access to the DB or entity APIs.

- Our long-term goals are to minimize these and convert them to PHPUnit

Testing interactions between services

Ideally your tests would construct your own isolated object graphs for direct manipulation, but this isn't always possible.

If your test relies on Elgg's Service Provider (`_elgg_services()` returns a `Elgg\Di\ServiceProvider`), realize that it maintains a singleton instance for most services it hands out, and many services keep their own local references to these services as well.

Due to these local references, replacing services on the SP within a test often will not have the desired effect. Instead, you may need to use functionality baked into the services themselves:

- The `events` and `hooks` services have methods `backup()` and `restore()`.
- The `logger` service has methods `disable()` and `enable()`.

Jasmine Tests

Test files must be named `*Test.js` and should go in either `js/tests/` or next to their source files in `views/default/**.js`. Karma will automatically pick up on new `*Test.js` files and run those tests.

Test boilerplate

```
define(function(require) {
        var elgg = require('elgg');

        describe("This new test", function() {
                it("fails automatically", function() {
                        expect(true).toBe(false);
                });
        });
});
```

Running the tests

Elgg uses Karma with Jasmine to run JS unit tests.

You will need to have nodejs and npm installed.

First install all the development dependencies:

```
npm install
```

Run through the tests just once and then quit:

```
npm test
```

You can also run tests continuously during development so they run on each save:

```
karma start js/tests/karma.conf.js
```

Debugging JS tests

You can run the test suite inside Chrome dev tools:

```
npm run chrome
```

This will output a URL like `http://localhost:9876/`.

1. Open the URL in Chrome, and click «Debug».
2. Open Chrome dev tools and the Console tab.
3. Reload the page.

If you alter a test you'll have to quit Karma with `Ctrl-c` and restart it.

6.3.4 Coding best practices

Make your code easier to read, easier to maintain, and easier to debug. Consistent use of these guidelines means less guess work for developers, which means happier, more productive developers.

General coding

Don't Repeat Yourself

If you are copy-pasting code a significant amount of code, consider whether there's an opportunity to reduce duplication by introducing a function, an additional argument, a view, or a new component class.

E.g. If you find views that are identical except for a single value, refactor into a single view that takes an option.

Note: In a bugfix release, *some duplication is preferrable to refactoring*. Fix bugs in the simplest way possible and refactor to reduce duplication in the next minor release branch.

Embrace SOLID and GRASP

Use these principles for OO design to solve problems using loosely coupled components, and try to make all components and integration code testable.

Whitespace is free

Don't be afraid to use it to separate blocks of code. Use a single space to separate function params and string concatenation.

Variable names

Use self-documenting variable names. `$group_guids` is better than `$array`.

Avoid double-negatives. Prefer `$enable = true` to `$disable = false`.

Interface names

Use the pattern `Elgg\{Namespace}\{Name}`.

Do not include an `I` prefix or an `Interface` suffix.

We do not include any prefix or suffix so that we're encouraged to:

- name implementation classes more descriptively (the «default» name is taken).
- type-hint on interfaces, because that is the shortest, easiest thing to do.

Name implementations like `Elgg\{Namespace}\{Interface}\{Implementation}`.

Functions

Where possible, have functions/methods return a single type. Use empty values such as `array()`, `""`, or `0` to indicate no results.

Be careful where valid return values (like `"0"`) could be interpreted as empty.

Functions not throwing an exception on error should return `false` upon failure.

Functions returning only boolean should be prefaced with `is_` or `has_` (eg, `elgg_is_logged_in()`, `elgg_has_access_to_entity()`).

Ternary syntax

Acceptable only for single-line, non-embedded statements.

Minimize complexity

Minimize nested blocks and distinct execution paths through code. Use Return Early to reduce nesting levels and cognitive load when reading code.

Use comments effectively

Good comments describe the «why.» Good code describes the «how.» E.g.:

Bad:

```
// increment $i only when the entity is marked as active.
foreach ($entities as $entity) {
        if ($entity->active) {
                $i++;
        }
}
```

Good:

```
// find the next index for inserting a new active entity.
foreach ($entities as $entity) {
        if ($entity->active) {
                $i++;
        }
}
```

Always include a comment if it's not obvious that something must be done in a certain way. Other developers looking at the code should be discouraged from refactoring in a way that would break the code.

```
// Can't use empty()/boolean: "0" is a valid value
if ($str === '') {
    register_error(elgg_echo('foo:string_cannot_be_empty'));
    forward(REFERER);
}
```

Commit effectively

- Err on the side of atomic commits which are highly focused on changing one aspect of the system.
- Avoid mixing in unrelated changes or extensive whitespace changes. Commits with many changes are scary and make pull requests difficult to review.
- Use visual git tools to craft highly precise and readable diffs.

Include tests

When at all possible *include unit tests* for code you add or alter.

Keep bugfixes simple

Avoid the temptation to refactor code for a bugfix release. Doing so tends to introduce regressions, breaking functionality in what should be a stable release.

PHP guidelines

These are the required coding standards for Elgg core and all bundled plugins. Plugin developers are strongly encouraged to adopt these standards.

Developers should first read the PSR-2 Coding Standard Guide.

Elgg's standards extend PSR-2, but differ in the following ways:

- Indent using one tab character, not spaces.
- Opening braces for classes, methods, and functions must go on the same line.
- If a line reaches over 100 characters, consider refactoring (e.g. introduce variables).
- Compliance with PSR-1 is encouraged, but not strictly required.

Documentation

- Include PHPDoc comments on functions and classes (all methods; declared properties when appropriate), including types and descriptions of all parameters.
- In lists of `@param` declarations, the beginnings of variable names and descriptions must line up.
- Annotate classes, methods, properties, and functions with `@access private` unless they are intended for public use, are already of limited visibility, or are within a class already marked as private.
- Use `//` or `/* */` when commenting.
- Use only `//` comments inside function/method bodies.

Naming

- Use underscores to separate words in the names of functions, variables, and properties. Method names are camelCase.
- Names of functions for public use must begin with `elgg_`.
- All other function names must begin with `_elgg_`.
- Name globals and constants in `ALL_CAPS` (`ACCESS_FRIENDS`, `$CONFIG`).

Miscellaneous

For PHP requirements, see `composer.json`.

Do not use PHP shortcut tags `<?` or `<%`. It is OK to use `<?=` since it is always enabled as of PHP 5.4.

When creating strings with variables:

- use double-quoted strings
- wrap variables with braces only when necessary.

Bad (hard to read, misuse of quotes and {}s):

```
echo 'Hello, '.$name."!  How is your {$time_of_day}?";
```

Good:

```
echo "Hello, $name!  How is your $time_of_day?";
```

Remove trailing whitespace at the end of lines. An easy way to do this before you commit is to run `php .scripts/fix_style.php` from the installation root.

CSS guidelines

Use shorthand where possible

Bad:

```
background-color: #333333;
background-image:  url(...);
background-repeat:  repeat-x;
background-position:  left 10px;
padding: 2px 9px 2px 9px;
```

Good:

```
background: #333 url(...) repeat-x left 10px;
padding: 2px 9px;
```

Use hyphens, not underscores

Bad:

```
.example_class {}
```

Good:

```
.example-class {}
```

One property per line

Bad:

```
color: white;font-size: smaller;
```

Good:

```
color: white;
font-size: smaller;
```

Property declarations

These should be spaced like so: `property: value;`

Bad:

```
color:value;
color :value;
color : value;
```

Good:

```
color: value;
```

Vendor prefixes

- Group vendor-prefixes for the same property together
- Longest vendor-prefixed version first
- Always include non-vendor-prefixed version
- Put an extra newline between vendor-prefixed groups and other properties

Bad:

```
-moz-border-radius: 5px;
border: 1px solid #999999;
-webkit-border-radius: 5px;
width: auto;
```

Good:

```
border: 1px solid #999999;

-webkit-border-radius: 5px;
-moz-border-radius: 5px;
border-radius: 5px;

width: auto;
```

Group subproperties

Bad:

```
background-color: white;
color: #0054A7;
background-position: 2px -257px;
```

Good:

```
background-color: white;
background-position: 2px -257px;
color: #0054A7;
```

Javascript guidelines

Same formatting standards as PHP apply.

All functions should be in the `elgg` namespace.

Function expressions should end with a semi-colon.

```
elgg.ui.toggles = function(event) {
        event.preventDefault();
        $(target).slideToggle('medium');
};
```

6.3.5 Deprecating APIs

Occasionally functions and classes must be deprecated in favor of newer replacements. Since 3rd party plugin authors rely on a consistent API, backward compatibility must be maintained, but will not be maintained indefinitely as plugin authors are expected to properly update their plugins. In order to maintain backward compatibility, deprecated APIs will follow these guidelines:

- Minor version (1.x) that deprecates an API must include a wrapper function/class (or otherwise appropriate means) to maintain backward compatibility, including any bugs in the original function/class. This compatibility layer uses `elgg_deprecated_notice('...', '1.11')` to log that the function is deprecated.
- The next major revision (2.0) removes the compatibility layer. Any use of the deprecated API should be corrected before this.

6.4 Adding a Service to Elgg

The *services guide* has general information about using Elgg services.

To add a new service object to Elgg:

1. Annotate your class as `@access private`.
2. Open the class `Elgg\Di\ServiceProvider`.
3. Add a `@property-read` annotation for your service at the top. This allows IDEs and static code analyzers to understand the type of the property.
4. To the constructor, add code to tell the service provider what to return. See the class `Elgg\Di\DiContainer` for more information on how Elgg's DI container works.

At this point your service will be available from the service provider object, but will not yet be accessible to plugins.

6.4.1 Inject your dependencies

Design your class constructor to *ask for* the necessary dependencies rather than creating them or using `_elgg_services()`. The service provider's `setFactory()` method provides access to the service provider instance in your factory method.

Here's an example of a `foo` service factory, injecting the `config` and `db` services into the constructor:

```
// in Elgg\Di\ServiceProvider::__construct()

$this->setFactory('foo', function (ServiceProvider $c) {
    return new Elgg\FooService($c->config, $c->db);
});
```

The full list of internal services can be seen in the `@property-read` declarations at the top of `Elgg\Di\ServiceProvider`.

Advertencia: Avoid performing work in your service constructor, particularly if it requires database queries. Currently PHPUnit tests cannot perform them.

6.4.2 Making a service part of the public API

If your service is meant for use by plugin developers:

1. Make an interface `Elgg\Services\<Name>` that contains only those methods needed in the public API.
2. Have your service class implement that interface.
3. For methods that are in the interface, move the documentation to the interface. You can simply use `{@inheritdoc}` in the PHPDocs of the concrete class methods.
4. Document your service in `docs/guides/services.rst` (this file).
5. Open the PHPUnit test `Elgg\ApplicationTest` and add your service key to the `$names` array in `testServices()`.
6. Open the class `Elgg\Application`.
7. Add `@property-read` declaration to document your service, but use your **interface** as the type, *not* your service class name.
8. Add your service key to the array in the `$public_services` property, e.g. `'foo' => true,`

Now your service will be available via property access on the `Elgg\Application` instance:

```
// using the public foo service
$three = elgg()->foo->add(1, 2);
```

Nota: For examples, see the `config` service, including the interface `Elgg\Services\Config` and the concrete implementation `Elgg\Config`.

Service Life Cycle and Factories

By default, services registered on the service provider are «shared», meaning the service provider will store the created instance for the rest of the request, and serve that same instance to all who request the property.

If you need developers to be able to construct objects that are pre-wired to Elgg services, you may need to add a public factory method to `Elgg\Application`. Here's an example that returns a new instance using internal Elgg services:

```
public function createFoo($bar) {
    $logger = $this->services->logger;
    $db = $this->services->db;
```

(continué en la próxima página)

(proviene de la página anterior)

```
    return new Elgg\Foo($bar, $logger, $db);
}
```

6.5 Writing Documentation

New documentation should fit well with the rest of Elgg's docs.

Contents

6.5.1 Testing docs locally

Elgg has a grunt script that automatically builds the docs, opens them in a browser window, and automatically reloads as you make changes (the reload takes just a few seconds). You need npm and sphinx installed to be able to use these scripts.

```
cd path/to/elgg/
npm install
grunt
```

It's that easy! Grunt will continue running, watching the docs for changes and automatically rebuilding.

6.5.2 Follow the existing document organization

The current breakdown is not necessarily the One True Way to organize docs, but consistency is better than randomness.

intro/*

This is everything that brand new users need to know (installation, features, license, etc.)

admin/*

Guides for administrators. Task-oriented.

guides/*

API guides for plugin developers. Cookbook-style. Example heavy. Code snippet heavy. Broken down by services (actions, i18n, routing, db, etc.). This should only discuss the public API and its behavior, not implementation details or reasoning.

design/*

Design docs for people who want to get a better understanding of how/why core is built the way it is. This should discuss internal implementation details of the various services, what tradeoffs were made, and the reasoning behind the final decision. Should be useful for people who want to contribute and for communication b/w core devs.

contribute/*

Contributors guides for the various ways people can participate in the project.

appendix/*

More detailed/meta/background information about the project (history, roadmap, etc.)

6.5.3 Use «Elgg» in a grammatically correct way

Elgg is not an acronym, so writing it in all caps (ELGG or E-LGG) is incorrect. Please don't do this.

In English, Elgg does not take an article when used as a noun. Here are some examples to emulate:

- "I'm using Elgg to run my website"
- "Install Elgg to get your community online"

When used as an adjective, the article applies to the main noun, so you should use one. For example:

- «Go to the Elgg community website to get help.»
- «I built an Elgg-based network yesterday»

This advice may not apply in languages other than English.

6.5.4 Avoid first person pronouns

Refer to the reader as "you." Do not include yourself in the normal narrative.

Before:

> When we're done installing Elgg, we'll look for some plugins!

After:

> When you're done installing Elgg, look for some plugins!

To refer to yourself (avoid this if possible), use your name and write in the third person. This clarifies to future readers/editors whose opinions are being expressed.

Before:

> I think the best way to do X is to use Y.

After:

Evan thinks the best way to do X is to use Y.

6.5.5 Eliminate fluff

Before:

> If you want to use a third-party javascript library within the Elgg framework, you should take care to call the `elgg_register_js` function to register it.

After:

> To use a third-party javascript library, call `elgg_register_js` to register it.

6.5.6 Prefer absolute dates over relative ones

It is not easy to tell when a particular sentence or paragraph was written, so relative dates quickly become meaningless. Absolute dates also give the reader a good indication of whether a project has been abandoned, or whether some advice might be out of date.

Before:

> Recently the foo was barred. Soon, the baz will be barred too.

After:

> Recently (as of September 2013), the foo was barred. The baz is expected to be barred by October 2013.

6.5.7 Do not remind the reader to contribute

Focus on addressing only the topic at hand. Constant solicitation for free work is annoying and makes the project look needy. If people want to contribute to the project, they can visit the contributor guide.

6.6 Internationalizing documentation

When you change documentation, remember to update the documentation translation templates before you commit:

```
cd docs/
make gettext
```

For more information, see http://sphinx-doc.org/latest/intl.html#translating-with-sphinx-intl

6.7 Becoming a Financial Supporter

All funds raised via the Elgg supporters network go directly into:

- Elgg core development
- Infrastructure provision (elgg.org, github, etc.)

It is a great way to help with Elgg development!

6.7.1 Benefits

For only $50 per year for individuals or $150 per year for organizations, you can get listed as a supporter on our supporters page. Elgg supporters are listed there unless they request not to be.

Supporters are able to put this official logo on their site if they wish:

6.7.2 Disclaimer

We operate a no refund policy on supporter subscriptions. If you would like to withdraw your support, go to PayPal and cancel your subscription. You will not be billed the following year.

Being an Elgg Supporter does not give an individual or organization the right to impersonate, trade as or imply they are connected to the Elgg project. They can, however, mention that they support the Elgg project.

If you have any questions about this disclaimer, email info@elgg.org.

We reserve the right to remove or refuse a listing without any prior warning at our complete discretion. There is no refund policy.

If there is no obvious use of Elgg, your site will be linked to with «nofollow» set.

6.7.3 Sign up

If you would like to become an Elgg supporter:

- read the *disclaimer* above
- on the supporters page, subscribe via PayPal
- send an email to info@elgg.org with:
  - the date you subscribed
  - your name (and organization name, if applicable)
  - your website
  - your Elgg community profile

Once all the details have been received, we will add you to the appropriate list. Thanks for your support!

6.8 Release Process Workflow

Release a new version of Elgg.

This is the process the core team follows for making a new Elgg release. We have published this information in the spirit of openness, and to streamline onboarding of new team members.

Contents

6.8.1 Requirements

- SSH access to elgg.org
- Commit access to http://github.com/Elgg/Elgg
- Admin access to https://elgg.org/
- Access to Twitter account
- Node.js and NPM installed
- Sphinx installed (`easy_install sphinx && easy_install sphinx-intl`)
- Transifex client installed (`easy_install transifex-client`)
- Transifex account with access to Elgg project

6.8.2 Update composer dependencies

Since Elgg 2.3, `composer.lock` is committed to the repository. Therefore, if any of the composer dependencies require an update, run `composer update` on the corresponding branch and make a pull request with an updated `composer.lock` file. This will run the test suite and ensure that new dependencies do not break the build.

6.8.3 Merge commits up from lower branches

Determine the LTS branch (currently 1.12). We need to merge any new commits there up through the other branches.

For each branch

Check out the branch, make sure it's up to date, and make a new work branch with the merge. E.g. here we're merging 1.12 commits into 2.0:

```
git checkout 2.0
git pull
git checkout -b merge112
git merge 1.12
```

Nota: If already up-to-date (no commits to merge), we can stop here for this branch.

If there are conflicts, resolve them, `git add .`, and `git merge`.

Make a PR for the branch and wait for automated tests and approval by other dev(s).

```
git push -u my_fork merge112
```

Once merged, we would repeat the process to merge 2.0 commits into 2.1.

6.8.4 First new stable minor/major release

Update the *Support policy* to include the new minor/major release date and fill in the blanks for the previous release.

6.8.5 Prepare the release

Bring your local git clone up to date.

Merge latest commits up from lowest supported branch.

Visit `https://github.com/Elgg/Elgg/compare/<new>...<old>` and submit the PR if there is anything that needs to be merged up.

Install the prerequisites:

```
npm install elgg-conventional-changelog
easy_install sphinx
easy_install sphinx-intl
easy_install transifex-client
```

Nota: On Windows you need to run these command in a console with admin privileges

Run the `release.php` script. For example, to release 1.12.5:

```
git checkout 1.12
php .scripts/release.php 1.12.5
```

This creates a `release-1.12.5` branch in your local repo.

Next, manually browse to the `/admin/settings/basic` page and verify it loads. If it does not, a language file from Transifex may have a PHP syntax error. Fix the error and amend your commit with the new file:

```
# only necessary if you fixed a language file
git add .
git commit --amend
```

Next, submit a PR via GitHub for automated testing and approval by another developer:

```
git push your-remote-fork release-1.12.5
```

6.8.6 Tag the release

Once approved and merged, tag the release:

```
git checkout release-${version}
git tag -a ${version} -m'Elgg ${version}'
git push --tags origin release-${version}
```

Or create a release on GitHub

- Goto releases
- Click "Draft a new release"
- Enter the version
- Select the correct branch (eg 1.12 for a 1.12.x release, 2.3 for a 2.3.x release, etc)
- Set the release title as "Elgg {version}"
- Paste the CHANGELOG.md part related to this release in the description

Some final administration

- Mark GitHub release milestones as completed
- Move unresolved tickets in released milestones to later milestones

6.8.7 Update the website

- ssh to elgg.org
- Clone https://github.com/Elgg/elgg-scripts

Build zip package

Use `elgg-scripts/build/elgg-starter-project.sh` to generate the .zip file. Run without arguments to see usage.

Nota: If this is your first time on the server building a release run `composer global require "fxp/composer-asset-plugin:^1.2.0"`. This will make sure you can download bower-assets during the build process.

```
    # login as user deploy
    sudo -su deploy

# regular release
./elgg-starter-project.sh master 2.0.4 /var/www/www.elgg.org/download/

# MIT release
./elgg-starter-project.sh master 2.0.4-mit /var/www/www.elgg.org/download/
```

- Verify that `vendor/elgg/elgg/composer.json` in the zip file has the expected version.

- If not, make sure GitHub has the release tag, and that the starter project has a compatible `elgg/elgg` item in the composer requires list.

Building 1.x zip packages

Use `elgg-scripts/build/build.sh` to generate the .zip file. Run without arguments to see usage.

```
# regular release
./build.sh 1.12.5 1.12.5 /var/www/www.elgg.org/download/

# MIT release
./build.sh 1.12.5 1.12.5-mit /var/www/www.elgg.org/download/
```

Update elgg.org download page

- Clone https://github.com/Elgg/community
- Add the new version to `classes/Elgg/Releases.php`
- Commit and push the changes
- Update the plugin on www.elgg.org

```
composer update elgg/community
```

Update elgg.org

- Clone https://github.com/Elgg/www.elgg.org
- Change the required Elgg version in `composer.json`
- Update vendors

```
composer update
```

- Commit and push the changes
- Pull to live site

```
cd /var/www/www.elgg.org && sudo su deploy && git pull
```

- Update dependencies

```
composer install --no-dev --prefer-dist --optimize-autoloader
```

- **Go to community admin panel**
    - Flush APC cache
    - Run upgrade

6.8.8 Make the announcement

This should be the very last thing you do.

1. Open `https://github.com/Elgg/Elgg/blob/<tag>/CHANGELOG.md` and copy the contents for that version
2. Sign in at https://elgg.org/blog and compose a new blog with a summary
3. Copy in the CHANGELOG contents, clear formatting, and manually remove the SVG anchors
4. Add tags `release` and `elgg2.x` where x is whatever branch is being released
5. Tweet from the elgg Twitter account

CAPÍTULO 7

Apéndice

Información variada sobre el proyecto.

7.1 FAQs and Other Troubleshooting

Below are some commonly asked questions about Elgg.

Contents

7.1.1 General

Ver también:

Getting Help

«Plugin cannot start and has been deactivated» or «This plugin is invalid»

This error is usually accompanied by more details explaining why the plugin is invalid. This is usually caused by an incorrectly installed plugin.

If you are installing a plugin called «test», there will be a test directory under mod. In that test directory there needs to be a start.php file: `/mod/test/start.php` and a manifest.xml file `/mod/test/manifest.xml`.

If these files do not exist, it could be caused by:

- installing a plugin to the wrong directory
- creating a directory under /mod that does not contain a plugin
- a bad ftp transfer
- unzipping a plugin into an extra directory (myplugin.zip unzips to `myplugin/myplugin`)

If you are on a Unix-based host and the files exist in the correct directory, check the permissions. Elgg must have read access to the files and read + execute access on the directories.

White Page (WSOD)

A blank, white page (often called a «white screen of death») means there is a PHP syntax error. There are a few possible causes

- corrupted file - try transfering the code again to your server
- a call to a php module that was not loaded - this can happen after you install a plugin that requires a specific module.
- bad plugin - not all plugins have been written to the same quality so you should be careful which ones you install.

To find where the error is occurring, change the .htaccess file to display errors to the browser. Set display_errors to 1 and load the same page again. You should see a PHP error in your browser. Change the setting back once you've resolved the problem.

Nota: If you are using the Developer's Tools plugin, go to its settings page and make sure you have «Display fatal PHP errors» enabled.

If the white screen is due to a bad plugin, remove the latest plugins that you have installed by deleting their directories and then reload the page.

Nota: You can temporarily disable all plugins by creating an empty file at `mod/disabled`. You can then disable the offending module via the administrator tools panel.

If you are getting a WSOD when performing an action, like logging in or posting a blog, but there are no error messages, it's most likely caused by non-printable characters in plugin code. Check the plugin for white spaces/new lines characters after finishing php tag (`?>`) and remove them.

Page not found

If you have recently installed your Elgg site, the most likely cause for a page not found error is that `mod_rewrite` is not setup correctly on your server. There is information in the *Install Troubleshooting* page on fixing this. The second most likely cause is that your site url in your database is incorrect.

If you've been running your site for a while and suddenly start getting page not found errors, you need to ask yourself what has changed. Have you added any plugins? Did you change your server configuration?

To debug a page not found error:

- Confirm that the link leading to the missing page is correct. If not, how is that link being generated?
- Confirm that the `.htaccess` rewrite rules are being picked up.

Login token mismatch

If you have to log in twice to your site and the error message after the first attempt says there was a token mismatch error, the URL in Elgg's settings does not match the URL used to access it. The most common cause for this is adding or removing the «www» when accessing the site. For example, www.elgg.org vs elgg.org. This causes a problem with session handling because of the way that web browsers save cookies.

To fix this, you can add rewrite rules. To redirect from www.elgg.org to elgg.org in Apache, the rules might look like:

```
RewriteCond %{HTTP_HOST} .
RewriteCond %{HTTP_HOST} !^elgg\.org
RewriteRule (.*) http://elgg.org/$1 [R=301,L]
```

Redirecting from non-www to www could look like this:

```
RewriteCond %{HTTP_HOST} ^elgg\.org
RewriteRule ^(.*)$ http://www.elgg.org/$1 [R=301,L]
```

If you don't know how to configure rewrite rules, ask your host for more information.

Form is missing __token or __ts fields

All Elgg actions require a security token, and this error occurs when that token is missing. This is either a problem with your server configuration or with a 3rd party plugin.

If you experience this on a new installation, make sure that your server is properly configured and your rewrite rules are correct. If you experience this on an upgrade, make sure you have updated your rewrite rules either in .htaccess (Apache) or in the server configuration.

If you are experiencing this, disable all 3rd party plugins and try again. Very old plugins for Elgg don't use security tokens. If the problem goes away when plugins are disabled, it's due to a plugin that should be updated by its author.

Maintenance mode

To take your site temporarily offline, go to Administration -> Utilities -> Maintenance Mode. Complete the form and hit save to disable your site for everyone except admin users.

Missing email

If your users are reporting that validation emails are not showing up, have them check their spam folder. It is possible that the emails coming from your server are being marked as spam. This depends on many factors such as whether your hosting provider has a problem with spammers, how your PHP mail configuration is set up, what mail transport agent your server is using, or your host limiting the number of email that you can send in an hour.

If no one gets email at all, it is quite likely your server is not configured properly for email. Your server needs a program to send email (called a Mail Transfer Agent - MTA) and PHP must be configured to use the MTA.

To quickly check if PHP and an MTA are correctly configured, create a file on your server with the following content:

```
<?php
$address = "your_email@your_host.com";

$subject = 'Test email.';

$body = 'If you can read this, your email is working.';

echo "Attempting to email $address...<br />";

if (mail($address, $subject, $body)) {
        echo 'SUCCESS!  PHP successfully delivered email to your MTA.  If you don\'t␣
↪see the email in your inbox in a few minutes, there is a problem with your MTA.';
} else {
        echo 'ERROR!  PHP could not deliver email to your MTA.  Check that your PHP␣
↪settings are correct for your MTA and your MTA will deliver email.';
}
```

Be sure to replace «your_email@your_host.com» with your actual email address. Take care to keep quotes around it! When you access this page through your web browser, it will attempt to send a test email. This test will let you know that PHP and your MTA are correctly configured. If it fails–either you get an error or you never receive the email–you will need to do more investigating and possibly contact your service provider.

Fully configuring an MTA and PHP's email functionality is beyond the scope of this FAQ and you should search the Internet for more resources on this. Some basic information on php parameters can be found on PHP's site

Server logs

Most likely you are using Apache as your web server. Warnings and errors are written to a log by the web server and can be useful for debugging problems. You will commonly see two types of log files: access logs and error logs. Information from PHP and Elgg is written to the server error log.

- Linux – The error log is probably in /var/log/httpd or /var/log/apache2.
- Windows - It is probably inside your Apache directory.
- Mac OS - The error log is probably in /var/log/apache2/error_log

If you are using shared hosting without ssh access, your hosting provider may provide a mechanism for obtaining access to your server logs. You will need to ask them about this.

How does registration work?

With a default setup, this is how registration works:

1. User fills out registration form and submits it
2. User account is created and disabled until validated
3. Email is sent to user with a link to validate the account
4. When a user clicks on the link, the account is validated
5. The user can now log in

Failures during this process include the user entering an incorrect email address, the validation email being marked as spam, or a user never bothering to validate the account.

User validation

By default, all users who self-register must validate their accounts through email. If a user has problems validating an account, you can validate users manually by going to Administration -> Users -> Unvalidated.

You can remove this requirement by deactivating the User Validation by Email plugin.

Nota: Removing validation has some consequences: There is no way to know that a user registered with a working email address, and it may leave you system open to spammers.

Manually add user

To manually add a user, under the Administer controls go to Users. There you will see a link title «Add new User». After you fill out the information and submit the form, the new user will receive an email with username and password and a reminder to change the password.

Nota: Elgg does not force the user to change the password.

I'm making or just installed a new theme, but graphics or other elements aren't working

Make sure the theme is at the bottom of the plugin list.

Clear your browser cache and reload the page. To lighten the load on the server, Elgg instructs the browser to rarely load the CSS file. A new theme will completely change the CSS file and a refresh should cause the browser to request the CSS file again.

If you're building or modifying a theme, make sure you have disabled the simple and system caches. This can be done by enabling the Developer Tools plugin, then browsing to Administration -> Develop -> Settings. Once you're satisfied with the changes, enable the caches or performance will suffer.

Changing profile fields

Within the Administration settings of Elgg is a page for replacing the default profile fields. Elgg by default gives the administrator two choices:

- Use the default profile fields
- Replace the default with a set of custom profile fields

You cannot add new profile fields to the default ones. Adding a new profile field through the replace profile fields option clears the default ones. Before letting in users, it is best to determine what profile fields you want, what field types they should be, and the order they should appear. You cannot change the field type or order or delete fields after they have been created without wiping the entire profile blank.

More flexibility can be gained through plugins. There is at least two plugins on the community site that enable you to have more control over profile fields. The Profile Manager plugin has become quite popular in the Elgg community. It lets you add new profile fields whenever you want, change the order, group profile fields, and add them to registration.

Changing registration

The registration process can be changed through a plugin. Everything about registration can be changed: the look and feel, different registration fields, additional validation of the fields, additional steps and so on. These types of changes require some basic knowledge of HTML, CSS, PHP.

Another option is to use the Profile Manager plugin that lets you add fields to both user profiles and the registration form.

Create the plugin skeleton *Plugin skeleton*

Changing registration display Override the `account/forms/register` view

Changing the registration action handler You can write your own action to create the user's account

How do I change PHP settings using .htaccess?

You may want to change php settings in your `.htaccess` file. This is especially true if your hosting provider does not give you access to the server's `php.ini` file. The variables could be related to file upload size limits, security, session length, or any number of other php attributes. For examples of how to do this, see the PHP documentation on this.

HTTPS login turned on accidently

If you have turned on HTTPS login but do not have SSL configured, you are now locked out of your Elgg install. To turn off this configuration parameter, you will need to edit your database. Use a tool like phpMyAdmin to view your database. Select the `config` table and delete the row that has the name `https_login`.

Using a test site

It is recommended to always try out new releases or new plugins on a test site before running them on a production site (a site with actual users). The easiest way to do this is to maintain a separate install of Elgg with dummy accounts. When testing changes it is important to use dummy accounts that are not admins to test what your users will see.

A more realistic test is to mirror the content from your production site to your test site. Following the instructions for *duplicating a site*. Then make sure you prevent emails from being sent to your users. You could write a small plugin that redirects all email to your own account (be aware of plugins that include their own custom email sending code so you'll have to modify those plugins). After this is done you can view all of the content to make sure the upgrade or new plugin is functioning as desired and is not breaking anything. If this process sounds overwhelming, please stick with running a simple test site.

500 - Internal Server Error

What is it?

A **500 - Internal Server Error** means the web server experienced a problem serving a request.

Ver también:

The Wikipedia page on HTTP status codes

Possible causes

Web server configuration The most common cause for this is an incorrectly configured server. If you edited the `.htaccess` file and added something incorrect, Apache will send a 500 error.

Permissions on files It could also be a permissions problem on a file. Apache needs to be able to read Elgg's files. Using permissions 755 on directories and 644 on files will allow Apache to read the files.

When I upload a photo or change my profile picture I get a white screen

Most likely you don't have the PHP GD library installed or configured properly. You may need assistance from the administrator of your server.

CSS is missing

Wrong URL

Sometimes people install Elgg so that the base URL is `localhost` and then try to view the site using a hostname. In this case, the browser won't be able to load the CSS file. Try viewing the source of the web page and copying the link for the CSS file. Paste that into your browser. If you get a 404 error, it is likely this is your problem. You will need to change the base URL of your site.

Syntax error

Elgg stores its CSS as PHP code to provide flexibility and power. If there is a syntax error, the CSS file served to the browser may be blank. Disabling non-bundled plugins is the recommended first step.

Rewrite rules errors

A bad `.htaccess` file could also result in a 404 error when requesting the CSS file. This could happen when doing an upgrade and forgetting to also upgrade `.htaccess`.

Should I edit the database manually?

> **Advertencia:** No, you should never manually edit the database!

Will editing the database manually break my site?

Yes.

Can I add extra fields to tables in the database?

(AKA: I don't understand the Elgg *data model* so I'm going to add columns. Will you help?)

No, this is a bad idea. Learn the *data model* and you will see that unless it's a very specific and highly customized installation, you can do everything you need within Elgg's current data model.

I want to remove users. Can't I just delete them from the elgg_users_entity table?

No, it will corrupt your database. Delete them through the site.

I want to remove spam. Can't I just search and delete it from the elgg_objects_entity table?

No, it will corrupt your database. Delete it through the site.

Someone on the community site told me to edit the database manually. Should I?

Who was it? Is it someone experienced with Elgg, like one of the core developers or a well-known plugin author? Did he or she give you clear and specific instructions on what to edit? If you don't know who it is, or if you can't understand or aren't comfortable following the instructions, do not edit the database manually.

I know PHP and MySQL and have a legitimate reason to edit the database. Is it okay to manually edit the database?

Make sure you understand Elgg's *data model* and schema first. Make a backup, edit carefully, then test copiously.

Internet Explorer (IE) login problem

Canonical URL

IE does not like working with sites that use both http://example.org and http://www.example.org. It stores multiple cookies and this causes problems. Best to only use one base URL. For details on how to do this see Login token mismatch error.

Chrome Frame

Using the chrome frame within IE can break the login process.

Emails don't support non-Latin characters

In order to support non-Latin characters, (such as Cyrillic or Chinese) Elgg requires multibyte string support to be compiled into PHP.

On many installs (e.g. Debian & Ubuntu) this is turned on by default. If it is not, you need to turn it on (or recompile PHP to include it). To check whether your server supports multibyte strings, check phpinfo.

Session length

Session length is controlled by your php configuration. You will first need to locate your `php.ini` file. In that file will be several session variables. A complete list and what they do can be found in the php manual.

File is missing an owner

There are three causes for this error. You could have an entity in your database that has an `owner_guid` of `0`. This should be extremely rare and may only occur if your database/server crashes during a write operation.

The second cause would be an entity where the owner no longer exists. This could occur if a plugin is turned off that was involved in the creation of the entity and then the owner is deleted but the delete operation failed (because the plugin is turned off). If you can figure out entity is causing this, look in your `entities` table and change the `owner_guid` to your own and then you can delete the entity through Elgg.

Advertencia: Reed the section «Should I edit the database manually?». Be very carefull when editing the database directly. It can break your site. **Always** make a backup before doing this.

The third cause is a user not having a username. This also indicates a database problem as this should not be possible. If it does occur, you could see this error when viewing a list of users (such as with the Members plugin). To fix, check your `users_entity` table for users without a username and if so, create a fake a username for that person. You should probably then delete the user through Elgg.

Fixes

Database Validator plugin will check your database for these causes and provide an option to fix them. Be sure to backup the database before you try the fix option.

No images

If profile images, group images, or other files have stopped working on your site it is likely due to a misconfiguration, especially if you have migrated to a new server.

These are the most common misconfigurations that cause images and other files to stop working.

Wrong path for data directory

Make sure the data directory's path is correct in the Site Administration admin area. It should have a trailing slash.

Wrong permissions on the data directory

Check the permissions for the data directory. The data directory should be readable and writeable by the web server user.

Different timezone

Nota: This only applies to Elgg versions before 1.9

If you migrated servers or upgraded PHP, check that PHP's timezone settings are the same between the old and the new. If you cannot or don't want to change the system-wide `php.ini` file, you can put the following at the top of `settings.php`:

```
date_default_timezone_set('MY_TIME_ZONE');
```

Where `MY_TIME_ZONE` is a valid PHP timezone.

Migrated installation with new data directory location

If you migrated an installation and need to change your data directory path, be sure to update the SQL for the filestore location as documented in the *Duplicate Installation* instructions.

Deprecation warnings

If you are seeing many deprecation warnings that say things like

```
Deprecated in 1.7: extend_view() was deprecated by elgg_extend_view()!
```

then you are using a plugin that was written for an older version of Elgg. This means the plugin is using functions that are scheduled to be removed in a future version of Elgg. You can ask the plugin developer if the plugin will be updated or you can update the plugin yourself. If neither of those are likely to happen, you should not use that plugin.

Javascript not working

If the user hover menu stops working or you cannot dismiss system messages, that means JavaScript is broken on your site. This usually due to a plugin having bad JavaScript code. You should find the plugin causing the problem and disable it. You can do this be disabling non-bundled plugins one at a time until the problem goes away. Another approach is disabling all non-bundled plugins and then enabling them one by one until the problem occurs again.

Most web browsers will give you a hint as to what is breaking the JavaScript code. They often have a console for JavaScript errors or an advanced mode for displaying errors. Once you see the error message, you may have an easier time locating the problem.

7.1.2 Security

Is upgrade.php a security concern?

Upgrade.php is a file used to run code and database upgrades. It is in the root of the directory and doesn't require a logged in account to access. On a fully upgraded site, running the file will only reset the caches and exit, so this is not a security concern.

If you are still concerned, you can either delete, move, or change permissions on the file until you need to upgrade.

Should I delete install.php?

This file is used to install Elgg and doesn't need to be deleted. The file checks if Elgg is already installed and forwards the user to the front page if it is.

Filtering

Filtering is used in Elgg to make XSS attacks more difficult. The purpose of the filtering is to remove Javascript and other dangerous input from users.

Filtering is performed through the function `filter_tags()`. This function takes in a string and returns a filtered string. It triggers a *validate*, *input* *plugin hook*. By default Elgg comes with the htmLawed filtering code as a plugin. Developers can drop in any additional or replacement filtering code as a plugin.

The `filter_tags()` function is called on any user input as long as the input is obtained through a call to `get_input()`. If for some reason a developer did not want to perform the default filtering on some user input, the `get_input()` function has a parameter for turning off filtering.

7.1.3 Development

What should I use to edit php code?

There are two main options: text editor or integrated development environment (IDE).

Text Editor

If you are new to software development or do not have much experience with IDEs, using a text editor will get you up and running the quickest. At a minimum, you will want one that does syntax highlighting to make the code easier to read. If you think you might submit patches to the bug tracker, you will want to make sure that your text editor does not change line endings. If you are using Windows, Notepad++ is a good choice. If you are on a Mac, TextWrangler is a popular choice. You could also give TextMate a try.

Integrated Development Environment

An IDE does just what its name implies: it includes a set of tools that you would normally use separately. Most IDEs will include source code control which will allow you to directly commit and update your code from your cvs repository. It may have an FTP client built into it to make the transfer of files to a remote server easier. It will have syntax checking to catch errors before you try to execute the code on a server.

The two most popular free IDEs for PHP developers are Eclipse and NetBeans. Eclipse has two different plugins for working with PHP code: PDT and PHPEclipse.

I don't like the wording of something in Elgg. How do I change it?

The best way to do this is with a plugin.

Create the plugin skeleton

Plugin skeleton

Locate the string that you want to change

All the strings that a user sees should be in the `/languages` directory or in a plugin's languages directory (`/mod/<plugin name>/languages`). This is done so that it is easy to change what language Elgg uses. For more information on this see the developer documentation on *Internacionalización* .

To find the string use `grep` or a text editor that provides searching through files to locate the string. (A good text editor for Windows is Notepad++) Let's say we want to change the string «Add friend» to «Make a new friend». The `grep` command to find this string would be `grep -r "Add friend" *`. Using Notepad++ , you would use the «Find

in files» command. You would search for the string, set the filter to `*.php`, set the directory to the base directory of Elgg, and make sure it searches all subdirectories. You might want to set it to be case sensitive also.

You should locate the string «Add friend» in `/languages/en.php`. You should see something like this in the file:

```
'friend:add' => "Add friend",
```

This means every time Elgg sees `friend:add` it replaces it with «Add friend». We want to change the definition of `friend:add`.

Override the string

To override this definition, we will add a languages file to the plugin that we built in the first step.

1. Create a new directory: `/mod/<your plugin name>/languages`
2. Create a file in that directory called `en.php`
3. Add these lines to that file

```
<?php

return array(
   'friend:add' => 'Make a new friend',
);
```

Make sure that you do not have any spaces or newlines before the `<?php`.

You're done now and should be able to enable the plugin and see the change. If you are override the language of a plugin, make sure your plugin is loaded after the one you are trying to modify. The loading order is determined in the Tools Administration page of the admin section. As you find more things that you'd like to change, you can keep adding them to this plugin.

How do I find the code that does x?

The best way to find the code that does something that you would like to change is to use `grep` or a similar search tool. If you do not have `grep` as a part of your operating system, you will want to install a grep tool or use a text-editor/IDE that has good searching in files. Notepad++ is a good choice for Windows users. Eclipse with PHP and NetBeans are good choices for any platform.

String Example

Let's say that you want to find where the *Log In* box code is located. A string from the *Log In* box that should be fairly unique is `Remember me`. `Grep` for that string. You will find that it is only used in the `en.php` file in the `/languages` directory. There it is used to define the *Internacionalización* string `user:persistent`. `Grep` for that string now. You will find it in two places: the same `en.php` language file and in `/views/default/forms/login.php`. The latter defines the html code that makes up the *Log In* box.

Action Example

Let's say that you want to find the code that is run when a user clicks on the *Save* button when arranging widgets on a profile page. View the Profile page for a test user. Use Firebug to drill down through the html of the page until you come to the action of the edit widgets form. You'll see the url from the base is `action/widgets/move`.

Grep on widgets/move and two files are returned. One is the JavaScript code for the widgets : /js/lib/ui.widgets.js. The other one, /engine/lib/widgets.php, is where the action is registered using elgg_register_action('widgets/reorder'). You may not be familiar with that function in which case, you should look it up at the API reference. Do a search on the function and it returns the documentation on the function. This tells you that the action is in the default location since a file location was not specified. The default location for actions is /actions so you will find the file at /actions/widgets/move.php.

Debug mode

During the installation process you might have noticed a checkbox that controlled whether debug mode was turned on or off. This setting can also be changed on the Site Administration page. Debug mode writes a lot of extra data to your php log. For example, when running in this mode every query to the database is written to your logs. It may be useful for debugging a problem though it can produce an overwhelming amount of data that may not be related to the problem at all. You may want to experiment with this mode to understand what it does, but make sure you run Elgg in normal mode on a production server.

Advertencia: Because of the amount of data being logged, don't enable this on a production server as it can fill up the log files really quick.

What goes into the log in debug mode?

- All database queries
- Database query profiling
- Page generation time
- Number of queries per page
- List of plugin language files
- Additional errors/warnings compared to normal mode (it's very rare for these types of errors to be related to any problem that you might be having)

What does the data look like?

```
[07-Mar-2009 14:27:20] Query cache invalidated
[07-Mar-2009 14:27:20] ** GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggentities where guid=1 and ( (1 = 1)  and␣
↪enabled='yes') results cached
[07-Mar-2009 14:27:20] SELECT guid from elggsites_entity where guid = 1 results cached
[07-Mar-2009 14:27:20] Query cache invalidated
[07-Mar-2009 14:27:20] ** GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggentities where guid=1 and ( (1 = 1)  and␣
↪enabled='yes') results cached
[07-Mar-2009 14:27:20] ** GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggentities where guid=1 and ( (1 = 1)  and␣
↪enabled='yes') results returned from cache
[07-Mar-2009 14:27:20] ** Sub part of GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggsites_entity where guid=1 results cached
[07-Mar-2009 14:27:20] Query cache invalidated
[07-Mar-2009 14:27:20] DEBUG: 2009-03-07 14:27:20 (MST): "Undefined index:  user" in␣
↪file /var/www/elgg/engine/lib/elgglib.php (line 62)
```

(continué en la próxima página)

(proviene de la página anterior)

```
[07-Mar-2009 14:27:20] DEBUG: 2009-03-07 14:27:20 (MST): "Undefined index:  pass" in␣
↪file /var/www/elgg/engine/lib/elgglib.php (line 62)
[07-Mar-2009 14:27:20] ****************** DB PROFILING ********************
[07-Mar-2009 14:27:20] 1 times: 'SELECT * from elggdatalists'
[07-Mar-2009 14:27:20] 1 times: 'SELECT * from elggentities where guid=1 and ( ␣
↪(access_id in (2) or (owner_guid = -1) or (access_id = 0 and owner_guid = -1)) and␣
↪enabled='yes')'
...
[07-Mar-2009 14:27:20] 2 times: 'update elggmetadata set access_id = 2 where entity_
↪guid = 1'
[07-Mar-2009 14:27:20] 1 times: 'UPDATE elggentities set owner_guid='0', access_id='2
↪', container_guid='0', time_updated='1236461868' WHERE guid=1'
[07-Mar-2009 14:27:20] 1 times: 'SELECT guid from elggsites_entity where guid = 1'
[07-Mar-2009 14:27:20] 1 times: 'UPDATE elggsites_entity set name='3124/944',␣
↪description='', url='http://example.org/' where guid=1'
[07-Mar-2009 14:27:20] 1 times: 'UPDATE elggusers_entity set prev_last_action = last_
↪action, last_action = 1236461868 where guid = 2'
[07-Mar-2009 14:27:20] DB Queries for this page: 56
[07-Mar-2009 14:27:20] ***********************************************
[07-Mar-2009 14:27:20] Page /action/admin/site/update_basic generated in 0.
↪36997294426 seconds
```

What events are triggered on every page load?

There are 5 *Elgg events* that are triggered on every page load:

1. boot, system
2. plugins_boot, system
3. init, system
4. pagesetup, system (deprecated)
5. shutdown, system

The *boot*, *system* event is triggered before the plugins get loaded. There does not appear to be any difference between the timing of the next two events: *plugins_boot*, *system* and *init*, *system* so plugins tend to use *init*, *system*. This event is triggered in `Elgg\Application::bootCore`. The *pagesetup*, *system* event is thrown the first time `elgg_view()` is called. Some pages like the default `index.php` do not call `elgg_view()` so it is not triggered for them. The *shutdown*, *system* event is triggered after the page has been sent to the requester and is handled through the PHP function `register_shutdown_function()`.

There are *other events* that are triggered by the Elgg core but they happen occasionally (such as when a user logs in).

What variables are reserved by Elgg?

- `$CONFIG`
- `$vars`
- `$autofeed`
- `$_GET['action']`/`$_POST['action']`
- `$viewtype`

Copy a plugin

There are many questions asked about how to copy a plugin. Let's say you want to copy the `blog` plugin in order to run one plugin called `blog` and another called `poetry`. This is not difficult but it does require a lot of work. You would need to

- change the directory name
- change the names of every function (having two functions causes PHP to crash)
- change the name of every view (so as not to override the views on the original plugin)
- change any data model subtypes
- change the language file
- change anything else that was specific to the original plugin

---

Nota: If you are trying to clone the `groups` plugin, you will have the additional difficulty that the group plugin does not set a subtype.

7.1.4 General

Ver también:

Getting Help

«Plugin cannot start and has been deactivated» or «This plugin is invalid»

This error is usually accompanied by more details explaining why the plugin is invalid. This is usually caused by an incorrectly installed plugin.

If you are installing a plugin called «test», there will be a test directory under mod. In that test directory there needs to be a start.php file: `/mod/test/start.php` and a manifest.xml file `/mod/test/manifest.xml`.

If these files do not exist, it could be caused by:

- installing a plugin to the wrong directory
- creating a directory under /mod that does not contain a plugin
- a bad ftp transfer
- unzipping a plugin into an extra directory (myplugin.zip unzips to `myplugin/myplugin`)

If you are on a Unix-based host and the files exist in the correct directory, check the permissions. Elgg must have read access to the files and read + execute access on the directories.

White Page (WSOD)

A blank, white page (often called a «white screen of death») means there is a PHP syntax error. There are a few possible causes

- corrupted file - try transfering the code again to your server
- a call to a php module that was not loaded - this can happen after you install a plugin that requires a specific module.

- bad plugin - not all plugins have been written to the same quality so you should be careful which ones you install.

To find where the error is occurring, change the .htaccess file to display errors to the browser. Set display_errors to 1 and load the same page again. You should see a PHP error in your browser. Change the setting back once you've resolved the problem.

Nota: If you are using the Developer's Tools plugin, go to its settings page and make sure you have «Display fatal PHP errors» enabled.

If the white screen is due to a bad plugin, remove the latest plugins that you have installed by deleting their directories and then reload the page.

Nota: You can temporarily disable all plugins by creating an empty file at `mod/disabled`. You can then disable the offending module via the administrator tools panel.

If you are getting a WSOD when performing an action, like logging in or posting a blog, but there are no error messages, it's most likely caused by non-printable characters in plugin code. Check the plugin for white spaces/new lines characters after finishing php tag (`?>`) and remove them.

Page not found

If you have recently installed your Elgg site, the most likely cause for a page not found error is that `mod_rewrite` is not setup correctly on your server. There is information in the *Install Troubleshooting* page on fixing this. The second most likely cause is that your site url in your database is incorrect.

If you've been running your site for a while and suddenly start getting page not found errors, you need to ask yourself what has changed. Have you added any plugins? Did you change your server configuration?

To debug a page not found error:

- Confirm that the link leading to the missing page is correct. If not, how is that link being generated?
- Confirm that the `.htaccess` rewrite rules are being picked up.

Login token mismatch

If you have to log in twice to your site and the error message after the first attempt says there was a token mismatch error, the URL in Elgg's settings does not match the URL used to access it. The most common cause for this is adding or removing the «www» when accessing the site. For example, www.elgg.org vs elgg.org. This causes a problem with session handling because of the way that web browsers save cookies.

To fix this, you can add rewrite rules. To redirect from www.elgg.org to elgg.org in Apache, the rules might look like:

```
RewriteCond %{HTTP_HOST} .
RewriteCond %{HTTP_HOST} !^elgg\.org
RewriteRule (.*) http://elgg.org/$1 [R=301,L]
```

Redirecting from non-www to www could look like this:

```
RewriteCond %{HTTP_HOST} ^elgg\.org
RewriteRule ^(.*)$ http://www.elgg.org/$1 [R=301,L]
```

If you don't know how to configure rewrite rules, ask your host for more information.

Form is missing __token or __ts fields

All Elgg actions require a security token, and this error occurs when that token is missing. This is either a problem with your server configuration or with a 3rd party plugin.

If you experience this on a new installation, make sure that your server is properly configured and your rewrite rules are correct. If you experience this on an upgrade, make sure you have updated your rewrite rules either in .htaccess (Apache) or in the server configuration.

If you are experiencing this, disable all 3rd party plugins and try again. Very old plugins for Elgg don't use security tokens. If the problem goes away when plugins are disabled, it's due to a plugin that should be updated by its author.

Maintenance mode

To take your site temporarily offline, go to Administration -> Utilities -> Maintenance Mode. Complete the form and hit save to disable your site for everyone except admin users.

Missing email

If your users are reporting that validation emails are not showing up, have them check their spam folder. It is possible that the emails coming from your server are being marked as spam. This depends on many factors such as whether your hosting provider has a problem with spammers, how your PHP mail configuration is set up, what mail transport agent your server is using, or your host limiting the number of email that you can send in an hour.

If no one gets email at all, it is quite likely your server is not configured properly for email. Your server needs a program to send email (called a Mail Transfer Agent - MTA) and PHP must be configured to use the MTA.

To quickly check if PHP and an MTA are correctly configured, create a file on your server with the following content:

```
<?php
$address = "your_email@your_host.com";

$subject = 'Test email.';

$body = 'If you can read this, your email is working.';

echo "Attempting to email $address...<br />";

if (mail($address, $subject, $body)) {
        echo 'SUCCESS!  PHP successfully delivered email to your MTA.  If you don\'t␣
↪see the email in your inbox in a few minutes, there is a problem with your MTA.';
} else {
        echo 'ERROR!  PHP could not deliver email to your MTA.  Check that your PHP␣
↪settings are correct for your MTA and your MTA will deliver email.';
}
```

Be sure to replace «your_email@your_host.com» with your actual email address. Take care to keep quotes around it! When you access this page through your web browser, it will attempt to send a test email. This test will let you know that PHP and your MTA are correctly configured. If it fails–either you get an error or you never receive the email–you will need to do more investigating and possibly contact your service provider.

Fully configuring an MTA and PHP's email functionality is beyond the scope of this FAQ and you should search the Internet for more resources on this. Some basic information on php parameters can be found on PHP's site

Server logs

Most likely you are using Apache as your web server. Warnings and errors are written to a log by the web server and can be useful for debugging problems. You will commonly see two types of log files: access logs and error logs. Information from PHP and Elgg is written to the server error log.

- Linux – The error log is probably in /var/log/httpd or /var/log/apache2.
- Windows - It is probably inside your Apache directory.
- Mac OS - The error log is probably in /var/log/apache2/error_log

If you are using shared hosting without ssh access, your hosting provider may provide a mechanism for obtaining access to your server logs. You will need to ask them about this.

How does registration work?

With a default setup, this is how registration works:

1. User fills out registration form and submits it
2. User account is created and disabled until validated
3. Email is sent to user with a link to validate the account
4. When a user clicks on the link, the account is validated
5. The user can now log in

Failures during this process include the user entering an incorrect email address, the validation email being marked as spam, or a user never bothering to validate the account.

User validation

By default, all users who self-register must validate their accounts through email. If a user has problems validating an account, you can validate users manually by going to Administration -> Users -> Unvalidated.

You can remove this requirement by deactivating the User Validation by Email plugin.

Nota: Removing validation has some consequences: There is no way to know that a user registered with a working email address, and it may leave you system open to spammers.

Manually add user

To manually add a user, under the Administer controls go to Users. There you will see a link title «Add new User». After you fill out the information and submit the form, the new user will receive an email with username and password and a reminder to change the password.

Nota: Elgg does not force the user to change the password.

I'm making or just installed a new theme, but graphics or other elements aren't working

Make sure the theme is at the bottom of the plugin list.

Clear your browser cache and reload the page. To lighten the load on the server, Elgg instructs the browser to rarely load the CSS file. A new theme will completely change the CSS file and a refresh should cause the browser to request the CSS file again.

If you're building or modifying a theme, make sure you have disabled the simple and system caches. This can be done by enabling the Developer Tools plugin, then browsing to Administration -> Develop -> Settings. Once you're satisfied with the changes, enable the caches or performance will suffer.

Changing profile fields

Within the Administration settings of Elgg is a page for replacing the default profile fields. Elgg by default gives the administrator two choices:

- Use the default profile fields
- Replace the default with a set of custom profile fields

You cannot add new profile fields to the default ones. Adding a new profile field through the replace profile fields option clears the default ones. Before letting in users, it is best to determine what profile fields you want, what field types they should be, and the order they should appear. You cannot change the field type or order or delete fields after they have been created without wiping the entire profile blank.

More flexibility can be gained through plugins. There is at least two plugins on the community site that enable you to have more control over profile fields. The Profile Manager plugin has become quite popular in the Elgg community. It lets you add new profile fields whenever you want, change the order, group profile fields, and add them to registration.

Changing registration

The registration process can be changed through a plugin. Everything about registration can be changed: the look and feel, different registration fields, additional validation of the fields, additional steps and so on. These types of changes require some basic knowledge of HTML, CSS, PHP.

Another option is to use the Profile Manager plugin that lets you add fields to both user profiles and the registration form.

Create the plugin skeleton *Plugin skeleton*

Changing registration display Override the `account/forms/register` view

Changing the registration action handler You can write your own action to create the user's account

How do I change PHP settings using .htaccess?

You may want to change php settings in your `.htaccess` file. This is especially true if your hosting provider does not give you access to the server's `php.ini` file. The variables could be related to file upload size limits, security, session length, or any number of other php attributes. For examples of how to do this, see the PHP documentation on this.

HTTPS login turned on accidently

If you have turned on HTTPS login but do not have SSL configured, you are now locked out of your Elgg install. To turn off this configuration parameter, you will need to edit your database. Use a tool like phpMyAdmin to view your database. Select the `config` table and delete the row that has the name `https_login`.

Using a test site

It is recommended to always try out new releases or new plugins on a test site before running them on a production site (a site with actual users). The easiest way to do this is to maintain a separate install of Elgg with dummy accounts. When testing changes it is important to use dummy accounts that are not admins to test what your users will see.

A more realistic test is to mirror the content from your production site to your test site. Following the instructions for *duplicating a site*. Then make sure you prevent emails from being sent to your users. You could write a small plugin that redirects all email to your own account (be aware of plugins that include their own custom email sending code so you'll have to modify those plugins). After this is done you can view all of the content to make sure the upgrade or new plugin is functioning as desired and is not breaking anything. If this process sounds overwhelming, please stick with running a simple test site.

500 - Internal Server Error

What is it?

A **500 - Internal Server Error** means the web server experienced a problem serving a request.

Ver también:

The Wikipedia page on HTTP status codes

Possible causes

Web server configuration The most common cause for this is an incorrectly configured server. If you edited the `.htaccess` file and added something incorrect, Apache will send a 500 error.

Permissions on files It could also be a permissions problem on a file. Apache needs to be able to read Elgg's files. Using permissions 755 on directories and 644 on files will allow Apache to read the files.

When I upload a photo or change my profile picture I get a white screen

Most likely you don't have the PHP GD library installed or configured properly. You may need assistance from the administrator of your server.

CSS is missing

Wrong URL

Sometimes people install Elgg so that the base URL is `localhost` and then try to view the site using a hostname. In this case, the browser won't be able to load the CSS file. Try viewing the source of the web page and copying the link for the CSS file. Paste that into your browser. If you get a 404 error, it is likely this is your problem. You will need to change the base URL of your site.

Syntax error

Elgg stores its CSS as PHP code to provide flexibility and power. If there is a syntax error, the CSS file served to the browser may be blank. Disabling non-bundled plugins is the recommended first step.

Rewrite rules errors

A bad `.htaccess` file could also result in a 404 error when requesting the CSS file. This could happen when doing an upgrade and forgetting to also upgrade `.htaccess`.

Should I edit the database manually?

> **Advertencia:** No, you should never manually edit the database!

Will editing the database manually break my site?

Yes.

Can I add extra fields to tables in the database?

(AKA: I don't understand the Elgg *data model* so I'm going to add columns. Will you help?)

No, this is a bad idea. Learn the *data model* and you will see that unless it's a very specific and highly customized installation, you can do everything you need within Elgg's current data model.

I want to remove users. Can't I just delete them from the elgg_users_entity table?

No, it will corrupt your database. Delete them through the site.

I want to remove spam. Can't I just search and delete it from the elgg_objects_entity table?

No, it will corrupt your database. Delete it through the site.

Someone on the community site told me to edit the database manually. Should I?

Who was it? Is it someone experienced with Elgg, like one of the core developers or a well-known plugin author? Did he or she give you clear and specific instructions on what to edit? If you don't know who it is, or if you can't understand or aren't comfortable following the instructions, do not edit the database manually.

I know PHP and MySQL and have a legitimate reason to edit the database. Is it okay to manually edit the database?

Make sure you understand Elgg's *data model* and schema first. Make a backup, edit carefully, then test copiously.

Internet Explorer (IE) login problem

Canonical URL

IE does not like working with sites that use both http://example.org and http://www.example.org. It stores multiple cookies and this causes problems. Best to only use one base URL. For details on how to do this see Login token mismatch error.

Chrome Frame

Using the chrome frame within IE can break the login process.

Emails don't support non-Latin characters

In order to support non-Latin characters, (such as Cyrillic or Chinese) Elgg requires multibyte string support to be compiled into PHP.

On many installs (e.g. Debian & Ubuntu) this is turned on by default. If it is not, you need to turn it on (or recompile PHP to include it). To check whether your server supports multibyte strings, check phpinfo.

Session length

Session length is controlled by your php configuration. You will first need to locate your `php.ini` file. In that file will be several session variables. A complete list and what they do can be found in the php manual.

File is missing an owner

There are three causes for this error. You could have an entity in your database that has an `owner_guid` of `0`. This should be extremely rare and may only occur if your database/server crashes during a write operation.

The second cause would be an entity where the owner no longer exists. This could occur if a plugin is turned off that was involved in the creation of the entity and then the owner is deleted but the delete operation failed (because the plugin is turned off). If you can figure out entity is causing this, look in your `entities` table and change the `owner_guid` to your own and then you can delete the entity through Elgg.

> **Advertencia:** Reed the section «Should I edit the database manually?». Be very carefull when editing the database directly. It can break your site. **Always** make a backup before doing this.

The third cause is a user not having a username. This also indicates a database problem as this should not be possible. If it does occur, you could see this error when viewing a list of users (such as with the Members plugin). To fix, check your `users_entity` table for users without a username and if so, create a fake a username for that person. You should probably then delete the user through Elgg.

Fixes

Database Validator plugin will check your database for these causes and provide an option to fix them. Be sure to backup the database before you try the fix option.

No images

If profile images, group images, or other files have stopped working on your site it is likely due to a misconfiguration, especially if you have migrated to a new server.

These are the most common misconfigurations that cause images and other files to stop working.

Wrong path for data directory

Make sure the data directory's path is correct in the Site Administration admin area. It should have a trailing slash.

Wrong permissions on the data directory

Check the permissions for the data directory. The data directory should be readable and writeable by the web server user.

Different timezone

Nota: This only applies to Elgg versions before 1.9

If you migrated servers or upgraded PHP, check that PHP's timezone settings are the same between the old and the new. If you cannot or don't want to change the system-wide `php.ini` file, you can put the following at the top of `settings.php`:

```
date_default_timezone_set('MY_TIME_ZONE');
```

Where `MY_TIME_ZONE` is a valid PHP timezone.

Migrated installation with new data directory location

If you migrated an installation and need to change your data directory path, be sure to update the SQL for the filestore location as documented in the *Duplicate Installation* instructions.

Deprecation warnings

If you are seeing many deprecation warnings that say things like

```
Deprecated in 1.7: extend_view() was deprecated by elgg_extend_view()!
```

then you are using a plugin that was written for an older version of Elgg. This means the plugin is using functions that are scheduled to be removed in a future version of Elgg. You can ask the plugin developer if the plugin will be updated or you can update the plugin yourself. If neither of those are likely to happen, you should not use that plugin.

Javascript not working

If the user hover menu stops working or you cannot dismiss system messages, that means JavaScript is broken on your site. This usually due to a plugin having bad JavaScript code. You should find the plugin causing the problem

and disable it. You can do this be disabling non-bundled plugins one at a time until the problem goes away. Another approach is disabling all non-bundled plugins and then enabling them one by one until the problem occurs again.

Most web browsers will give you a hint as to what is breaking the JavaScript code. They often have a console for JavaScript errors or an advanced mode for displaying errors. Once you see the error message, you may have an easier time locating the problem.

7.1.5 Security

Is upgrade.php a security concern?

Upgrade.php is a file used to run code and database upgrades. It is in the root of the directory and doesn't require a logged in account to access. On a fully upgraded site, running the file will only reset the caches and exit, so this is not a security concern.

If you are still concerned, you can either delete, move, or change permissions on the file until you need to upgrade.

Should I delete install.php?

This file is used to install Elgg and doesn't need to be deleted. The file checks if Elgg is already installed and forwards the user to the front page if it is.

Filtering

Filtering is used in Elgg to make XSS attacks more difficult. The purpose of the filtering is to remove Javascript and other dangerous input from users.

Filtering is performed through the function `filter_tags()`. This function takes in a string and returns a filtered string. It triggers a *validate*, *input plugin hook*. By default Elgg comes with the htmLawed filtering code as a plugin. Developers can drop in any additional or replacement filtering code as a plugin.

The `filter_tags()` function is called on any user input as long as the input is obtained through a call to `get_input()`. If for some reason a developer did not want to perform the default filtering on some user input, the `get_input()` function has a parameter for turning off filtering.

7.1.6 Development

What should I use to edit php code?

There are two main options: text editor or integrated development environment (IDE).

Text Editor

If you are new to software development or do not have much experience with IDEs, using a text editor will get you up and running the quickest. At a minimum, you will want one that does syntax highlighting to make the code easier to read. If you think you might submit patches to the bug tracker, you will want to make sure that your text editor does not change line endings. If you are using Windows, Notepad++ is a good choice. If you are on a Mac, TextWrangler is a popular choice. You could also give TextMate a try.

Integrated Development Environment

An IDE does just what its name implies: it includes a set of tools that you would normally use separately. Most IDEs will include source code control which will allow you to directly commit and update your code from your cvs repository. It may have an FTP client built into it to make the transfer of files to a remote server easier. It will have syntax checking to catch errors before you try to execute the code on a server.

The two most popular free IDEs for PHP developers are Eclipse and NetBeans. Eclipse has two different plugins for working with PHP code: PDT and PHPEclipse.

I don't like the wording of something in Elgg. How do I change it?

The best way to do this is with a plugin.

Create the plugin skeleton

Plugin skeleton

Locate the string that you want to change

All the strings that a user sees should be in the `/languages` directory or in a plugin's languages directory (`/mod/<plugin name>/languages`). This is done so that it is easy to change what language Elgg uses. For more information on this see the developer documentation on *Internacionalización* .

To find the string use `grep` or a text editor that provides searching through files to locate the string. (A good text editor for Windows is Notepad++) Let's say we want to change the string «Add friend» to «Make a new friend». The `grep` command to find this string would be `grep -r "Add friend" *`. Using Notepad++ , you would use the «Find in files» command. You would search for the string, set the filter to `*.php`, set the directory to the base directory of Elgg, and make sure it searches all subdirectories. You might want to set it to be case sensitive also.

You should locate the string «Add friend» in `/languages/en.php`. You should see something like this in the file:

```
'friend:add' => "Add friend",
```

This means every time Elgg sees `friend:add` it replaces it with «Add friend». We want to change the definition of `friend:add`.

Override the string

To override this definition, we will add a languages file to the plugin that we built in the first step.

1. Create a new directory: `/mod/<your plugin name>/languages`
2. Create a file in that directory called `en.php`
3. Add these lines to that file

```
<?php

return array(
   'friend:add' => 'Make a new friend',
);
```

Make sure that you do not have any spaces or newlines before the `<?php`.

You're done now and should be able to enable the plugin and see the change. If you are override the language of a plugin, make sure your plugin is loaded after the one you are trying to modify. The loading order is determined in the Tools Administration page of the admin section. As you find more things that you'd like to change, you can keep adding them to this plugin.

How do I find the code that does x?

The best way to find the code that does something that you would like to change is to use `grep` or a similar search tool. If you do not have `grep` as a part of your operating system, you will want to install a grep tool or use a text-editor/IDE that has good searching in files. Notepad++ is a good choice for Windows users. Eclipse with PHP and NetBeans are good choices for any platform.

String Example

Let's say that you want to find where the *Log In* box code is located. A string from the *Log In* box that should be fairly unique is `Remember me`. `Grep` for that string. You will find that it is only used in the `en.php` file in the `/languages` directory. There it is used to define the *Internacionalización* string `user:persistent`. `Grep` for that string now. You will find it in two places: the same `en.php` language file and in `/views/default/forms/login.php`. The latter defines the html code that makes up the *Log In* box.

Action Example

Let's say that you want to find the code that is run when a user clicks on the *Save* button when arranging widgets on a profile page. View the Profile page for a test user. Use Firebug to drill down through the html of the page until you come to the action of the edit widgets form. You'll see the url from the base is `action/widgets/move`.

`Grep` on `widgets/move` and two files are returned. One is the JavaScript code for the widgets : `/js/lib/ui.widgets.js`. The other one, `/engine/lib/widgets.php`, is where the action is registered using `elgg_register_action('widgets/reorder')`. You may not be familiar with that function in which case, you should look it up at the API reference. Do a search on the function and it returns the documentation on the function. This tells you that the action is in the default location since a file location was not specified. The default location for actions is `/actions` so you will find the file at `/actions/widgets/move.php`.

Debug mode

During the installation process you might have noticed a checkbox that controlled whether debug mode was turned on or off. This setting can also be changed on the Site Administration page. Debug mode writes a lot of extra data to your php log. For example, when running in this mode every query to the database is written to your logs. It may be useful for debugging a problem though it can produce an overwhelming amount of data that may not be related to the problem at all. You may want to experiment with this mode to understand what it does, but make sure you run Elgg in normal mode on a production server.

> **Advertencia:** Because of the amount of data being logged, don't enable this on a production server as it can fill up the log files really quick.

What goes into the log in debug mode?

- All database queries
- Database query profiling
- Page generation time
- Number of queries per page
- List of plugin language files
- Additional errors/warnings compared to normal mode (it's very rare for these types of errors to be related to any problem that you might be having)

What does the data look like?

```
[07-Mar-2009 14:27:20] Query cache invalidated
[07-Mar-2009 14:27:20] ** GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggentities where guid=1 and ( (1 = 1)  and␣
↪enabled='yes') results cached
[07-Mar-2009 14:27:20] SELECT guid from elggsites_entity where guid = 1 results cached
[07-Mar-2009 14:27:20] Query cache invalidated
[07-Mar-2009 14:27:20] ** GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggentities where guid=1 and ( (1 = 1)  and␣
↪enabled='yes') results cached
[07-Mar-2009 14:27:20] ** GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggentities where guid=1 and ( (1 = 1)  and␣
↪enabled='yes') results returned from cache
[07-Mar-2009 14:27:20] ** Sub part of GUID:1 loaded from DB
[07-Mar-2009 14:27:20] SELECT * from elggsites_entity where guid=1 results cached
[07-Mar-2009 14:27:20] Query cache invalidated
[07-Mar-2009 14:27:20] DEBUG: 2009-03-07 14:27:20 (MST): "Undefined index:  user" in␣
↪file /var/www/elgg/engine/lib/elgglib.php (line 62)
[07-Mar-2009 14:27:20] DEBUG: 2009-03-07 14:27:20 (MST): "Undefined index:  pass" in␣
↪file /var/www/elgg/engine/lib/elgglib.php (line 62)
[07-Mar-2009 14:27:20] ***************** DB PROFILING ********************
[07-Mar-2009 14:27:20] 1 times: 'SELECT * from elggdatalists'
[07-Mar-2009 14:27:20] 1 times: 'SELECT * from elggentities where guid=1 and ( ␣
↪(access_id in (2) or (owner_guid = -1) or (access_id = 0 and owner_guid = -1)) and␣
↪enabled='yes')'
...
[07-Mar-2009 14:27:20] 2 times: 'update elggmetadata set access_id = 2 where entity_
↪guid = 1'
[07-Mar-2009 14:27:20] 1 times: 'UPDATE elggentities set owner_guid='0', access_id='2
↪', container_guid='0', time_updated='1236461868' WHERE guid=1'
[07-Mar-2009 14:27:20] 1 times: 'SELECT guid from elggsites_entity where guid = 1'
[07-Mar-2009 14:27:20] 1 times: 'UPDATE elggsites_entity set name='3124/944',␣
↪description='', url='http://example.org/' where guid=1'
[07-Mar-2009 14:27:20] 1 times: 'UPDATE elggusers_entity set prev_last_action = last_
↪action, last_action = 1236461868 where guid = 2'
[07-Mar-2009 14:27:20] DB Queries for this page: 56
[07-Mar-2009 14:27:20] ************************************************
[07-Mar-2009 14:27:20] Page /action/admin/site/update_basic generated in 0.
↪36997294426 seconds
```

What events are triggered on every page load?

There are 5 *Elgg events* that are triggered on every page load:

1. boot, system
2. plugins_boot, system
3. init, system
4. pagesetup, system (deprecated)
5. shutdown, system

The *boot*, *system* event is triggered before the plugins get loaded. There does not appear to be any difference between the timing of the next two events: *plugins_boot*, *system* and *init*, *system* so plugins tend to use *init*, *system*. This event is triggered in `Elgg\Application::bootCore`. The *pagesetup*, *system* event is thrown the first time `elgg_view()` is called. Some pages like the default `index.php` do not call `elgg_view()` so it is not triggered for them. The *shutdown*, *system* event is triggered after the page has been sent to the requester and is handled through the PHP function `register_shutdown_function()`.

There are *other events* that are triggered by the Elgg core but they happen occasionally (such as when a user logs in).

What variables are reserved by Elgg?

- `$CONFIG`
- `$vars`
- `$autofeed`
- `$_GET['action']`/`$_POST['action']`
- `$viewtype`

Copy a plugin

There are many questions asked about how to copy a plugin. Let's say you want to copy the `blog` plugin in order to run one plugin called `blog` and another called `poetry`. This is not difficult but it does require a lot of work. You would need to

- change the directory name
- change the names of every function (having two functions causes PHP to crash)
- change the name of every view (so as not to override the views on the original plugin)
- change any data model subtypes
- change the language file
- change anything else that was specific to the original plugin

---

Nota: If you are trying to clone the `groups` plugin, you will have the additional difficulty that the group plugin does not set a subtype.

7.2 Plan

¿En qué dirección va el proyecto? ¿Qué nuevas y emocionantes funcionalidades podemos esperar a corto plazo?

No publicamos planes detallados, pero los siguientes recursos le permitirán hacerse una idea de la dirección que estamos siguiendo:

- En nuestro grupo de retroalimentación y planificación tienen lugar las discusiones iniciales sobre aquello en lo que pensamos trabajar a continuación.
- Nuestros‘hitos en Github‘_ representan la dirección general para las versiones futuras de Elgg. Es lo más parecido que tenemos a un plan tradicional.
- Las solicitudes de integración de cambios de Github le indicarán en qué se está trabajando en estos momentos, pero nada es completamente seguro hasta que no se aceptan las solicitudes.
- Usamos el blog de desarrollo para publicar anuncios de funcionalidades que se han añadido recientemente en la rama de desarrollo. Se trata de la información más fiable disponible sobre nuevas funcionalidades que estarán disponibles en la siguiente versión.

7.2.1 Objetivos y valores

Tenemos una serie de objetivos y valores globales que afectan a la dirección de Elgg. Las mejoras deben promover en general estos valores para que las aceptemos.

Accesibilidad

Los sitios basados en Elgg debería poder utilizarlos cualquiera. Eso significa que lucharemos por hacer de Elgg una plataforma:

- Para cualquier dispositivo — fácil de usar desde móviles, tabletas, equipos de escritorio, etc.
- Para cualquier idioma — internacionalización, idiomas que se leen de derecha a izquierda, etc.
- Para cualquier capacidad — para utilizar mediante tacto, teclado, lectores de pantalla, etc.

Pruebas

Queremos **hacer innecesarias las pruebas manuales** por parte del equipo principal de desarrollo, de los creadores de complementos y de los administradores de sitios. Para ello promovemos y ayudamos a elaborar pruebas rápidas y automáticas para todos los aspectos de Elgg.

Consideramos que las API no están bien si obligan a los desarrolladores de complementos a escribir código que no se puede probar. Somos conscientes de que existen un gran número de casos en los que no se cumple con este principio, pero estamos trabajando en ello.

Esperamos que llegue el día en que los desarrolladores principales no necesiten hacer pruebas manuales para verificar que todo el código contribuido a Elgg está bien. De manera similar, nuestra visión incluye un mundo en el que los administradores de sitios pueden actualizar e instalar complementos nuevos con la confianza de que todo funciona correctamente en conjunto.

Pendiente: ¿otros objetivos o valores?

7.2.2 Preguntas frecuentes

¿Cuando se añadirá la funcionalidad «x»?

No podemos hacer promesas sobre cuándo se añadirán qué funcionalidades porque a Elgg sólo se le añaden nuevas funcionalidades cuando una persona está lo suficientemente motivada para trabajar en ellas y enviar una solicitud de integración de cambios. Lo único que podemos hacer es sugerirle que intente averiguar en qué funcionalidades han mostrado los desarrolladores existentes su deseo de trabajar.

The best way to ensure a feature gets implemented is to discuss it with the core team and implement it yourself. See our *Contributor Guides* guide if you're interested. We love new contributors!

Si está intentando decidir si usar o no Elgg, no se decida por Elgg en base a funcionalidades que aún no están disponibles. Evalúe Elgg por sus funcionalidades actuales. Lo más probable es que las funcionalidades futuras no estén listas para cuando usted las necesite.

¿Cuando publicáis la versión X.Y.Z?

La siguiente versión se publicará cuando el equipo principal de desarrollo considere que está lista y dispone de tiempo para prepararla. La información de http://github.com/Elgg/Elgg/issues/milestones debería darle una idea aproximada de los tiempos.

7.3 Política de publicaciones

Qué esperar al actualizar Elgg.

We adhere to semantic versioning.

Siga el blog para mantenerse informado de las últimas versiones publicadas.

7.3.1 Patch/Bugfix Releases (2.1.x)

Every two weeks.

Bugfix releases are made regularly to make sure Elgg stays stable, secure, and bug-free. The higher the third digit, the more tested and stable the release is.

Since bugfix release focus on fixing bugs and not making major changes, themes and plugins should work from bugfix release to bugfix release.

7.3.2 Minor/Feature Releases (2.x.0)

Every three months.

Whenever we introduce new features, we'll bump the middle version number. These releases aren't as mature as bugfix release, but are considered stable and useable.

We make every effort to be backward compatible in these releases, so plugins should work from minor release to minor release.

However, plugins might need to be updated to make use of the new features.

7.3.3 Major/Breaking Releases (x.0.0)

Every year.

Inevitably, improving Elgg requires breaking changes and a new major release is made. These releases are opportunities for the core team to make strategic, breaking changes to the underlying platform. Themes and plugins from older versions are not expected to work without modification on different major releases.

We may remove deprecated APIs, but we will not remove APIs without first deprecating them.

Elgg's dependencies may be upgraded by their major version or removed entirely. We will not remove any dependences before a major release, but we do not «deprecate» dependencies or issue any warnings before removing them.

Your package, plugin, or app should declare its own dependencies directly so that this does not cause a problem.

7.3.4 Alphas, Betas, and Release Candidates

Before major releases (and sometimes before feature releases), the core team will offer a pre-release version of Elgg to get some real-world testing and feedback on the release. These are meant for testing only and should not be used on a live site.

SemVer 2.0 does not define a particular meaning for pre-releases, but we approach alpha, beta, and rc releases with these general guidelines:

An `-alpha.X` pre-release means that there are still breaking changes planned, but the feature set of the release is frozen. No new features or breaking changes can be proposed for that release.

A `-beta.X` pre-release means that there are no known breaking changes left to be included, but there are known regressions or critical bugs left to fix.

An `-rc.X` pre-release means that there are no known regressions or critical bugs left to be fixed. This version could become the final stable version of Elgg if no new blockers are reported.

7.4 Support policy

As of Elgg 2.0, each minor release receives bug and security fixes only until the next minor release.

7.4.1 Long Term Support Releases

Within each major version, the last minor release is designated for long term support («LTS») and will receive bug fixes until the 2nd following major version release, and security fixes until the 3rd following major version release.

E.g. 1.12 is the last minor release within 1.x. It will receive bug fixes until 3.0 is released and security fixes until 4.0 is released.

When bugs are found, a good faith effort will be made to patch the LTS release, but **not all fixes will be back-ported.** E.g. some fixes may depend on new APIs, break backwards compatibility, or require significant refactoring. If a fix risks stability of the LTS branch, it will not be included.

Ver también:

Política de publicaciones

Below is a table outlining the specifics for each release (future dates are tentative):

Version	First stable release	Bug fixes through	Security fixes through
1.12 LTS	July 2015	**Until 3.0**	**Until 4.0**
2.0	December 2015	March 2016	
2.1	March 2016	June 2016	
2.2	June 2016	November 2016	
2.3 LTS	November 2016	**Until 4.0**	**Until 5.0**
3.0	December 2016		
4.0	December 2017		

7.5 Historia

El nombre viene de una ciudad suiza. También significa «alce» o «ratón» en danés.

Elgg's initial funding was by a company called Curverider Ltd, which was started by David Tosh and Ben Werdmuller. In 2010, Curverider was acquired by Thematic Networks and control of the open-source project was turned over to The Elgg Foundation. Today, Elgg is a community-driven open source project and has a variety of contributors and supporters.