eDirectory 11.0.00

Survival guide

What changed?

- Introducing Symfony 2.7 framework
- Partial directories restructure
- Language files
- Search engine
- Config files
- Composer insertion
- Templates system (themes)
- Javascript’s normalizing

What (Barely) DIDN’T change

- Site manager
- Sponsors
- Profile
- Database

Symfony 2.7

PHP MVC Framework

What is, what's for, how it works and why it has been chosen

Symfony 2.7

- Framework MVC (Model-View-Controller)

  - Model :

    Shapes every piece of information on the database

  - View :

    Organizes the way every information will be displayed

  - Controller :

    It is the bridge between the items above, obtaining information from the first, modeling it, and feeding the second.

Model

- A PHP-class representation of something that is shaped at the database

- It is used to ease the controller's info manipulation, as well as concentrate a big slice of the SQL commands that were scattered across the code previously

- Doctrine is the name of Symfony's library with that role

Entity example : Setting

The lines above the class definition determine the table which the entity will represent.

Within a class, before defining each propriety, the comment blocks (PHPDOC) defines what table column that given propiety represents, as well as some metadata about the propriety.

Important reminder: Every entity must be mapped ONLY ONCE to a table. Better saying, the entity-table bound must be 1 for 1.

```php
<?php

namespace ArcaSolutions\WebBundle\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * Setting
 *
 * @ORM\Table(name="Setting")
 * @ORM\Entity(repositoryClass="ArcaSolutions\WebBundle\Repository\SettingRepository")
 */
class Setting
{
    /**
     * @var string
     *
     * @ORM\Column(name="name", type="string", length=255, nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="IDENTITY")
     */
    private $name;

    /**
     * @var string
     *
     * @ORM\Column(name="value", type="text", nullable=false)
     */
    private $value;
```

Entities Usage

Here's an example of how to alter and insert new information into your database through entities.

```php
class DefaultController extends Controller
{
    public function rebuildLocationsAction()
    {
        $newSetting = new Setting();

        $newSetting->setName("sitemgr_name");
        $newSetting->setValue('Anakin');

        /* Salvando alterações */
        $objectManager = $this->get("doctrine")->getManager();
        $objectManager->persist($newSetting);
        $objectManager->flush();

        /* Escolhendo o repositório */
        $repository = $this->get("doctrine")->getRepository("WebBundle:Setting");
        /* Buscando uma setting já existente */
        $oldSetting = $repository->findOneBy(["name" => "sitemgr_email"]);
        /* Alterando o valor */
        $oldSetting->setValue('vader@deathstar.com');

        /* Salvando alterações */
        $objectManager = $this->get("doctrine")->getManager();
        $objectManager->flush();
    }
}
```

Entities Usage

Examples of how to obtain and delete information from the database

```php
class DefaultController extends Controller
{
    public function rebuildLocationsAction()
    {
        /* Escolhendo o repositório */
        $repository = $this->get("doctrine")->getRepository("WebBundle:Setting");
        /* Buscando por um determinado campo */
        $result = $repository->findOneBy(["name" => "elasticsearchRebuildLocations"]);

        /* Utilizando valores carregados do banco */
        $settingValue = $result->getValue();

        /* Removendo do banco */
        $objectManager = $this->get("doctrine")->getManager();
        $objectManager->remove($result);
        $objectManager->flush();
    }
}
```

View

- It’s the framework’s slice responsible for creating an information displaying template
- It’s basically the HTML master container
- The same way Doctrine handles the Model, on Symfony we use Twig to handle the View
- Twig is an intermediate language between PHP and HTML, easy to learn and more understandable for designers
- Twig is “Compiled” in time of execution and saved in cache. For that reason, a longer delay at the first time the template is rendered is perfectly normal.

Example of a “View” on Twig

```
{% block alerts %}
    {% for flashMessage in app.session.flashbag.get('notice') %}
        <div class="block-container">
            <div class="container">
                <div class="alert alert-{{ flashMessage.alert }} alert-dismissible" role="alert">
                    <button type="button" class="close" data-dismiss="alert">
                        <span aria-hidden="true">&times;</span><span class="sr-only">Close</span>
                    </button>
                    <strong>{{ flashMessage.title }}</strong> {{ flashMessage.message }}
                </div>
            </div>
        </div>
    {% endfor %}
{% endblock alerts %}
```

The code above creates an alert for each error message, working just like a “foreach”

Twig - Filters and functions

- Twig provides a massive variety of predefined filters and functions, all set and ready to go

- These functions can be found at the Twig's documentation

- This tool also allows creating new filters and custom functions, through codes named "Twig extension". The eDirectory has various custom extensions, for a vast realm of purposes(example: build the path to a certain image at the custom folder, format monetary values, add Javascript codes, etcetera)

- Twig also supports extend templates and specific blocks replacement on the page. This is used a lot on the newest version, for every page is inherited from "base.html.twig"

Twig - Filters and functions example

Function's example:

addJsFile(Custom) – Adds a JS file which will be included at the page's footer

```
{{ addJSFile("assets/js/lib/social-likes/social-likes.min.js") }}
{{ addJSFile("assets/js/modules/socialbuttons.js") }}
<div class="social-sharing social-likes_notext" data-zeroes="yes">
    <div class="share-facebook facebook"></div>
    <div class="share-facebook plusone"></div>
    <div class="share-twitter twitter"></div>
</div>
```

```
<div class="col-sm-8">
    {% if copyright_text -%}
        {# Custom text for copyrights -#}
        {{ copyright_text }}
    {% else -%}
        {# Default text for copyrights -#}
        {{ 'Copyright'|trans }} &copy; {{ 'now'|date('Y') }}
        {{ ' Arca Solutions, Inc.'|trans }}
        {{ 'All Rights Reserved.'|trans }}
    {% endif -%}
</div>
```

Filter's example:

trans – Translates a text's shard

date(custom) – formats a given date

Twig - related assets and functions

Images, CSS files and JS are located inside the “web/assets” folder

The **asset(“name”, “package”)** function is used to create a path to a determinate resource. This function accepts “package names” as second argument, which act as shortcuts to determinate paths. Here’s a list of all included the packages included in this version of eDirectory:

profile_images : custom/profile/
assets_images : assets/images/
domain_images : custom/domain_#/image_files/
domain_content : custom/domain_#/content_files/
domain_extrafiles : custom/domain_#/extra_files/

Usage example:

To include image ‘web/assets/images/bg-image.png’ ->

```
<img src="{{ asset("bg-image.png", "assets_images") }}">
```

Controller

- It's a PHP class that responds to a determinate route

- Routes are URL standards which are defined at routing.yml file, which will be discussed up ahead. Each URL standard points out to one Controller, which decides what to do as soon as it receives the applications control.

  Ex: www.demodirectory.com/listing will hand the execution to the listing's DefaultController.php, which will call the indexAction() method.

- The Controller's role is to process the needed information to feed the View with data that requires the Model to be displayed correctly

Control and Action example

```php
class DefaultController extends Controller
{
    ... Outras Actions ...

    public function allcategoriesAction()
    {
        $categories = $this->getDoctrine()
            ->getRepository('ListingBundle:ListingCategory')
            ->getAllParent();

        $data = [
            'categories' => $categories,
            'routing'    => $this->get("search.engine")->getModuleAlias("listing"),
        ];

        $response = $this->render('::modules/listing/all-categories.html.twig', $data);

        return $response;
    }

    ... Mais actions ...
}
```

Alright, but how does it all go together?

Basically, we've configured a ton of routes that end up in different controllers. Each Controller (which is a simple PHP class) decides what must happen in that page, making every needed request and alteration on the database through Entities at the Model and renders the proper View for that route, getting the needed info so it can show all the content that page must have.

Services

- Side classes with specific functionalities.
- Used to group functions that potentially will be used over and over again in other parts of the project.
- Accessed through the Container's get("name") method.
- Just like Twig's functions, there are native symfony services and custom eDirectory services. Both are accessed the same way.

Service example

The following service is used to get the content from the “CustomText” table according to its ID. The service’s name is configured at services.yml of the specific Bundle, as it can be seen below.

```
parameters:
    customtexthandler.service.class: ArcaSolutions\WebBundle\Services\CustomTextHandler

services:
    customtexthandler:
        class: %customtexthandler.service.class%
        arguments: ["@service_container"]
```

Service usage example:

```
$keywords = $container->get('customtexthandler')->get('header_keywords');
```

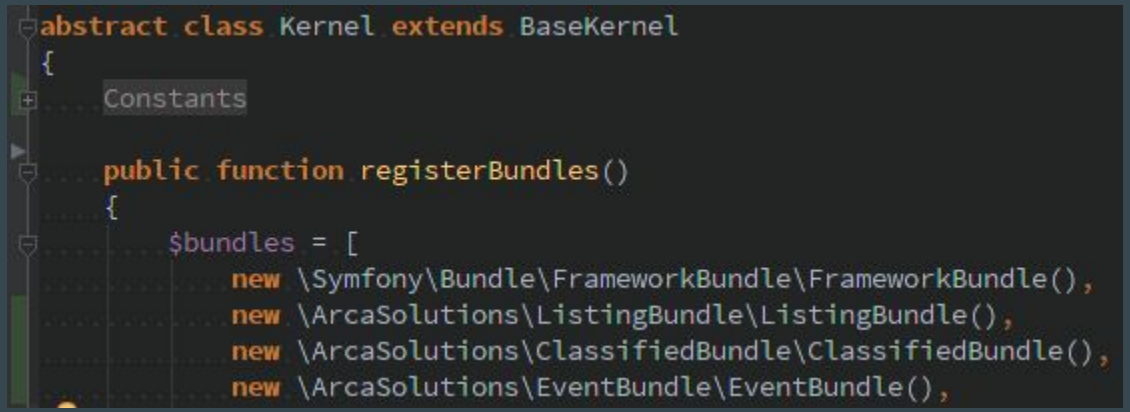
```
class CustomTextHandler
{
    /** @var Container ...*/
    private $container;

    function __construct($container)
    {
        $this->container = $container;
    }

    /** Retrieves the value of the custom text with the $name id ...*/
    public function get($name, $entity = false)
    {
        $return = null;

        if ($text = $this->container->get('doctrine')->getRepository("WebBundle:CustomText")->find($name)) {
            $return = $entity ? $text : $text->getValue();
        }

        return $return;
    }

    /** Adds or updates a CustomText entry ...*/
    public function add($name, $value)
    {
        $customText = $this->get($name, true);

        if (!$customText) {
            $customText = new CustomText();
            $customText->setName($name);
            $customText->setValue($value);
        }

        $em = $this->container->get("doctrine")->getManager();
        $em->persist($customText);
        $em->flush();
    }
}
```

Container

- This class is the main services and parameters manager
- Parameters are info defined at the *.yml files
- Instances the services just at the first time, turning them into a “Singleton” application afterwards
- The Container is available in every Controllers by default, although it can be “injected” into the services builder during its declaring at services.yml. This concept is called “Dependency Injection”

Bundles

- Are “Packages” of Controllers, Entities, Twigs & Services
- Contains all the files related to a determinate functionality
- Bundles are registered directly at the Kernel :



Why Symfony?

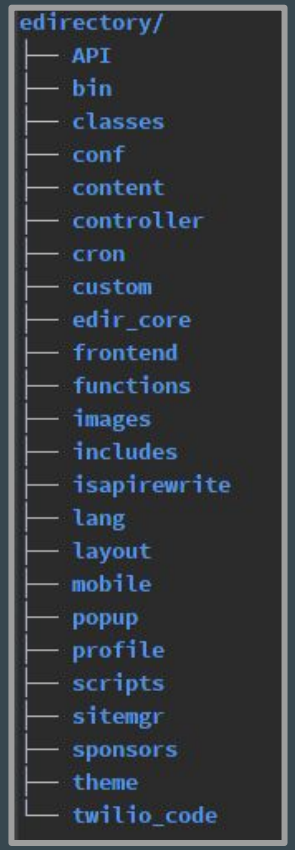
- It’s one of most used frameworks on the market, known for being robust and having a brand community and support
- Many of the other PHP frameworks have their origin at Symfony
- Its characteristics fit well with the projects must-have tools.
- Using a framework that considers the modern development models helps us in keeping our codes quality, eases adaptation for new developers and documentation, as well as being the first step to modern-up eDirectory

New directories structure

Files organization and Symfonys structure

How it was, how it became and why we needed the change

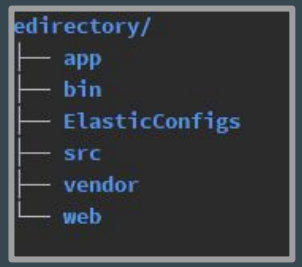
Previous directory structure



At the previous structure, all the folders were found at the server's root, which we'll call 'public_html' from now on.

Accessing the files before was direct, except for the front pages, which went through the old 'full_modrewrite.php', which identified the proper route and include the needed files in each section.

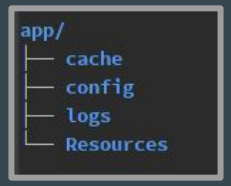
New Directory Structure



A new level before ‘public_html’ was added. This level contains all the codes and resources like configurations and translations of the new version

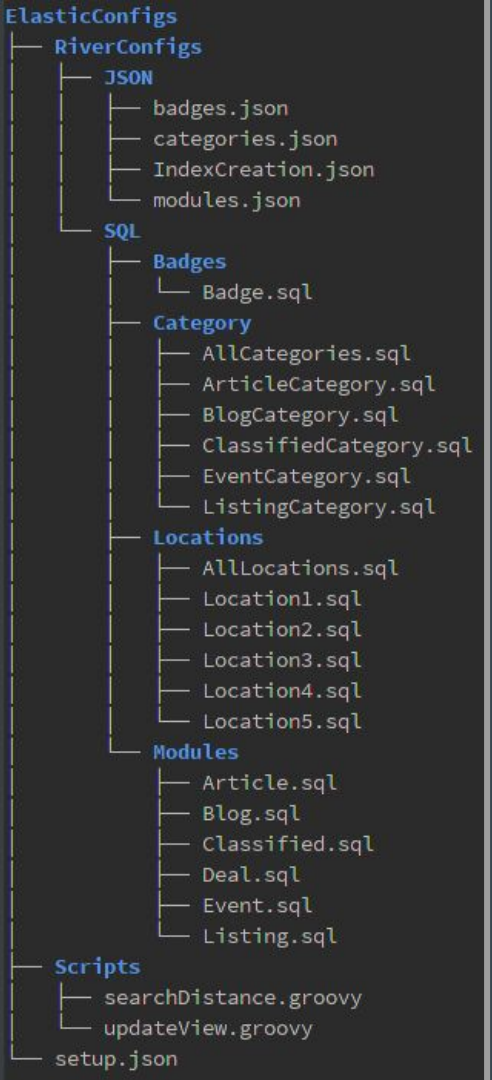
This folder that holds ‘public_html’ is safer, for its files and folders are not accessible directly through the browser

/app



Folder containing the Frameworks brain. Here, the main classes and configurations can be found. Contains:

- /app/cache
  - Configuration cache and Twig from every environment
- /app/config
  - Every configuration of the new version
  - The main routing file (routing.yml)
- /app/logs
  - Every log generated by the Framework
- /app/Resources
  - All the lang files
  - Theme files
  - Upgrade folder, where upgrade packages can be opened and executed
- Kernel
- Console



/ElasticConfigs

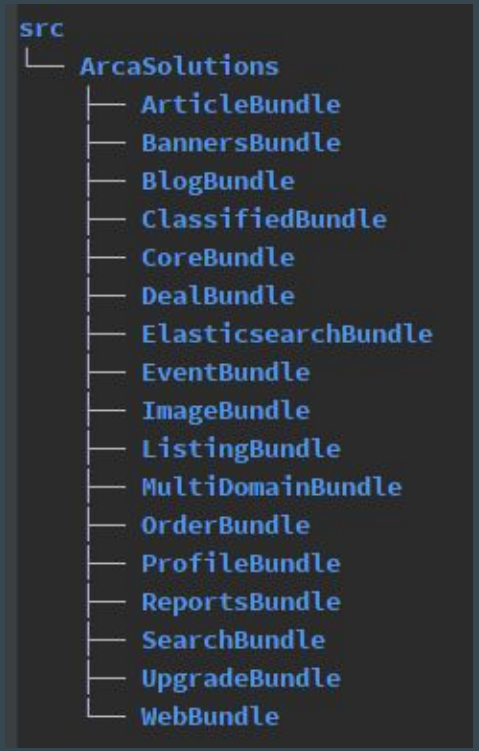
Directory containing all the MySQL and JSON code used by ElasticSearch during Synchronization.

Except the files located inside ElasticConfigs/Scripts, every other resource is used in the system. In case some SQL or JSON file is altered, those changes will propagate through the ElasticSearch and may be troublesome if not done carefully.

The SQL files content is used every time an item is saved, either at the sponsor's area or site manager. The query's result has the expected data shape for the ElasticSearch index. Such data shape is mapped in the file below:

ElasticConfigs/RiverConfigs/JSON/IndexCreation.json

(This process has be confirmed if still updated)

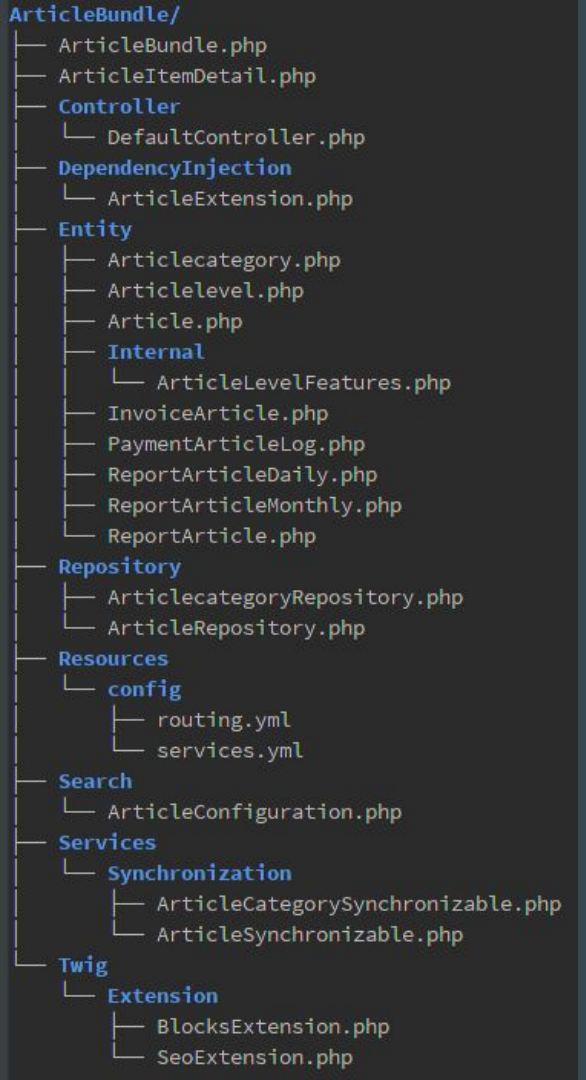


/src/ArcaSolutions

Folder that contains every Basecode team made Bundle. Each Bundle was projected to best bear all of a specific module's functionalities.

In the following pages, we'll further detail every Bundle we consider important. We'll explain its particularities, functionalities and how they interact with other Bundles.

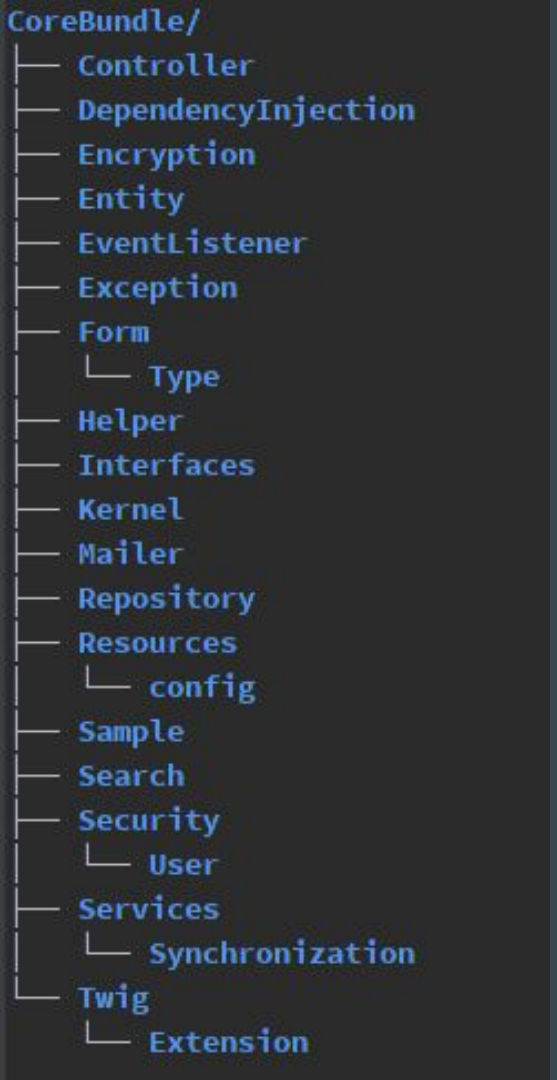
Not every Bundle will be detailed, for their functionalities, in a general context, is very alike.



The common in every Bundle

Basically, each module's Bundle has its specific Controllers, Entities, Services, Routes and Twig extensions. On the following example, we have the Articles Bundle:

- Controller: Every controller is placed here. In this case, the 'DefaultController.php' concentrates every action in this module. In other words, every URL-accessible code through routes defined at 'routing.yml' is in this class.
- Entity: Every relevant entity for the Article module, as well as some useful, non-mapped classes
- Repository: Entity-extending classes, providing custom functions that allow manipulating any data related to said entity.
- Resources: You'll find routing.yml and services.yml here.
    - routing.yml : Defines each and every route on this module, as well as how and which action of which controller each route will activate.
    - services.yml : Defines every service on the module, their names and what they need to be initialized.
- Search: Search related configurations. Self-explanatory.
- Services: Folder containing every service classes.
- Twig: Directory where the Twig extensions (filters, methods) can be found.

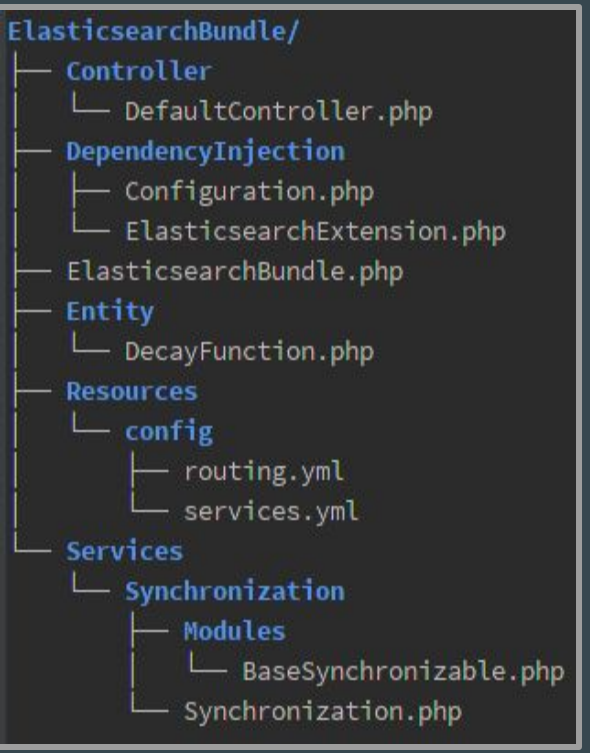


/src/ArcaSolutions/CoreBundle

The Core, MultiDomain and Web Bundle are the new eDirectory foundation. They bear support classes for other modules so they'll work correctly, as well as tons of utility classes like the "Utility" service, for example.

- Controller: Contains the MaintenanceController, which is called whenever the directory goes into Maintenance mode.
- Entity: EVERY mapped entity on the Main database
- EventListener: Holds every class that reacts to various Symfony's events
- Exception: Contains the directory's specific exceptions
- Kernel: Bears the eDirectory's custom Kernel, where every Bundle is registered

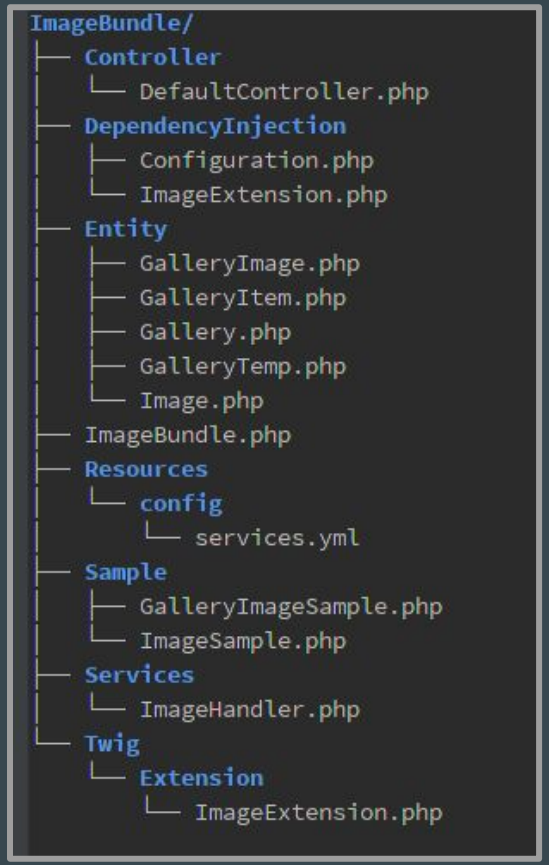
/src/ArcaSolutions/ElasticsearchBundle



This Bundle worries about MySQL's data synchronization with ElasticSeach's.

- Controller: Its DefaultController contains a hash encrypted route. If accessed with the "elasticsearchRebuildLocations" flag set to 1 at the Settings table at the Domain database, this action will rebuild the whole eDirectory locations index. This is used when a location level is enabled/disabled

- Services: Maybe the most important resource in this Bundle is the Sync service, located in the Synchronization.php class. That class is responsible for executing all the 'syncommands' whenever an item is saved.



/src/ArcaSolutions/ImageBundle

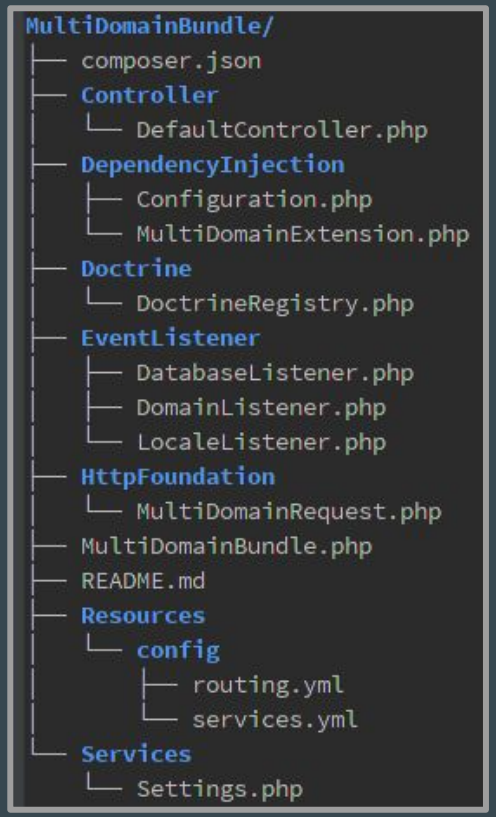
Contains entities related to galleries and images. Pretty self-explanatory

Its service ImageHandler allows it to obtain the path to any image from any instance from the Image class (do NOT confuse with class_Image.php)

This Bundle provides three Twig extensions which are extraordinary useful when searching for images. These are:

- imagePath(Image)
- imageProfile(AccountProfileContact)
- imageProfileByAccountId(Integer)

/src/ArcaSolutions/MultiDomainBundle



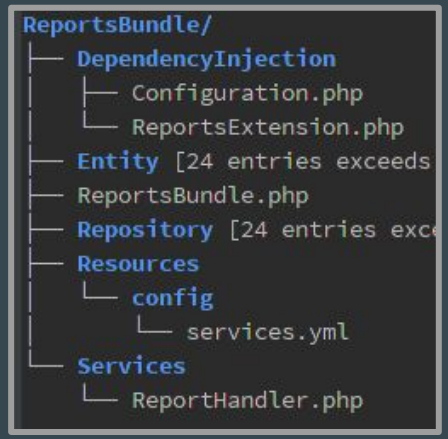
The MultiDomain Bundle is responsible for individual information maintenance in each domain, as well as intercept the Framework's initializing to inject the database information linked to said domain inside Doctrine.

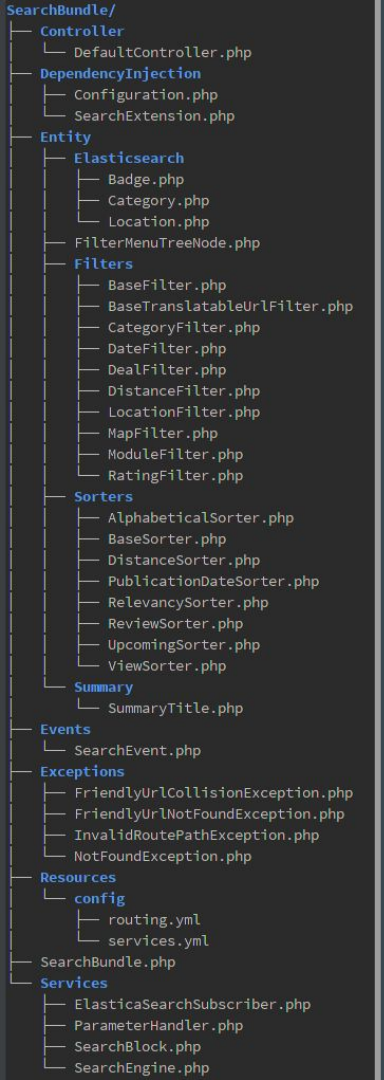
It's Settings ("multi_domain.information") service is the class to be used whenever you need to obtain specific information about the active domain, like for example the Domains ID, the path to its custom folder, its name, theme, database's name (Both MySQL and ElasticSearch)

/src/ArcaSolutions/ReportsBundle

This Bundle handles the reports created by the eDirectory newest version. That is done through its ReportHandler(“reporthandler”) service.

Besides, this Bundle holds all the entities that are somehow related to Reports(Hidden in the image for being too many)



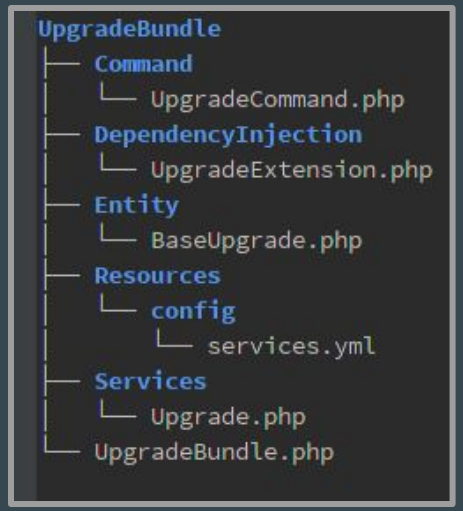


/src/ArcaSolutions/SearchBundle

The Search Bundle is responsible for every search related operation on eDirectory newest version. It was designed to be as modular and adaptive as possible.

- Controller: Its DefaultController contains the standard search actions, including the main search (searchAction).
- Entity:
  - Filters: Holds the classes that manage every search filters functionality
  - Sorters: Contains class that manage all the search orderers.
  - Summary: Bears the SummaryTitle class, responsible for building a “humanly readable” phrase which represents the content that is being searched.
- Events: Contains a container-like class for the search engine, which will be explained further ahead.
- Services:
  - ParameterHandler : Class responsible for obtaining information for the search from the URL
  - SearchEngine: Every search’s heart. This class coordinates every search procedure, including the Elasticsearch queries

/src/ArcaSolutions/UpgradeBundle

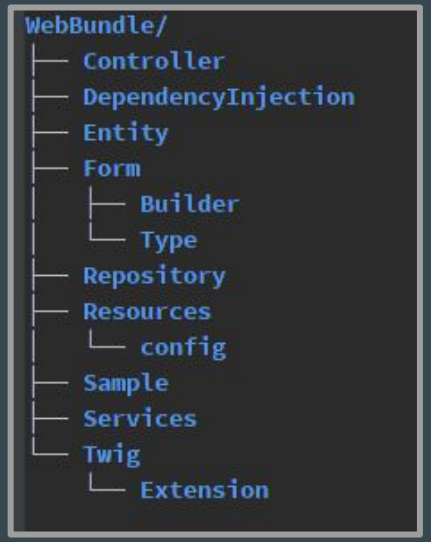


Responsible for locating and executing the upgrade patches. This Bundle is composed by a console command (php app/console edirectory:upgrade), the Upgrade('upgrade') service and the abstract class BaseUpgrade.

The command basically calls the service and provides the output(normally at the terminal)
The service researches the upgrade/ directory, searching for folders that contain SQL queries, which will be executed, and .php files that contain class that inherit the BaseUpgrade class. Those will be instantiated and have their Execute() method called

The BaseUpgrade class has methods which are useful for the upgrade process, like for example the elasticsearch rebuild and error log functions

/src/ArcaSolutions/WebBundle



It handles a ton of functionalities, including the controller which manages the homepage, a big slice of the Twig extensions and content management services, like LeadHandler, TimelineHandler & CustomTextHandler.

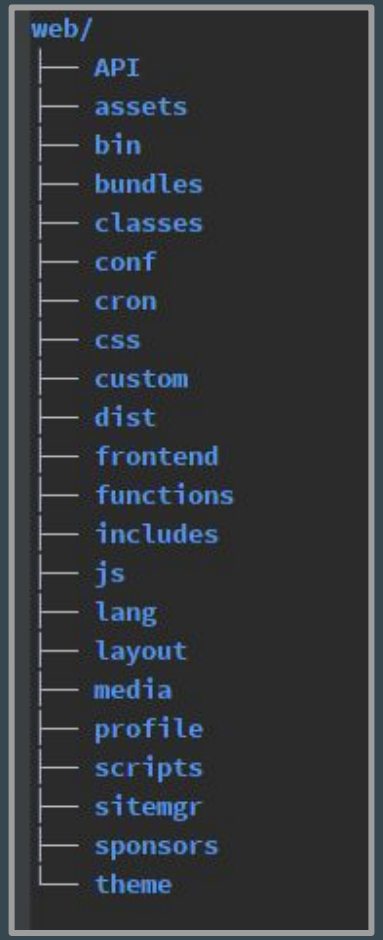
This Bundle also contains the mapping of all the entities in the Domain database which don't fit in no other Bundle (e.g.: Setting, FAQ, Review)

Among its multiple controllers, we have those which control the functionality of many sections, such as ContactUs, CustomPages, Enquire, Reviews & SendMail, as well as the main index, of course!

/vendor

This folder holds every default Bundle, which was not developed by our basecode team.

It is law; Thou shall never alter any code in this folder



/web

Dèja vu? Got the feeling you've encountered this folder before?

The web folder is the old 'public_html', which contains the old eDirectory structure. The Site Manager, Profile and Sponsors area are still at the old coding structure, and despite our plans of altering and leaving eDirectory entirely on Symfony's structure, that process will take long to be completed.

In other words, for this release only the Front was rebuild from the ground up. That doesn't mean that there were no changes in other parts of the code : A lot of stuff was removed and/or redone.

In this directory we also have the Assets folder, which contain all the CSS, JS and image files.

Language and Translation

XLIFF files

What, porquoi, & como?

Translation and language

Previously, every language was defined in a .php file (e.g.web/lang/en_us.php), which used constants to keep messages.

In this newer version, all those messages were moved to XLIFF files (.xliff or .xlf) which are located at app/Resources/translations. These files are basically XMLs which contain a node for each translation. Each node has embedded the original text and its translation, respectively at “source” and “target”.

```
<trans-unit id="912b26140b7aa062f072b8b0d335da705daa1477" resname=" Refine Search">
  <source>Refine Search</source>
  <target>Filtrar</target>
  <jms:reference-file line="51">blocks/filters/distance.html.twig</jms:reference-file>
</trans-unit>
```

Alright, but how does it work?

Instead of calling a constant all the time, in the newest eDirectory there are two ways of translating a text, depending of the context

- At Twig :

```
<h2 class="theme-title">
    {{ 'Featured Listings'|trans }}
    <a href="{{ path("/listing/") }}" class="view-more">{{ 'Explore all featured Listings'|trans }}</a>
</h2>
```

- Inside a Controller (Using the “translator” service):

```
/* Apply sorting */
$translatedSortParameterName = $this->container->get("translator")->trans("sort", [], "filters");
```

Translation domains

Another new feature is the translation domain.

Instead of grouping all the messages in a single file, in this version some translation were divided in various domains.

In sum, a “domain” is pretty much a neat name for “different file”. Lang used on the URLs and sorters names at results, for example, are allocated at filters.en.xliff. The default format uses the following rule:

domain.langinitials.xliff

Important notes

- Langs are compiled and cache is created out of them (app/cache/env/domain/translations). Sometimes this cache must be recreated so lang changes are effective. This can be done by deleting the cache folder of that domain or using the cache cleaning command (php app/console cache:clear --no-warmup)
- The file which concentrates the biggest slice of languages is messages.linguage.xliff
- There are tools to ease the translating process, but those are available in development environment only.

Themes

Reborn

Where have they gone and how they work?

Themes

In the newest version, we sought a way to leave themes as less dependent from codes as possible and give wider freedom for designers to modify them and create their own without having to understand complex PHP codes.

The solution was to move everything layout related to a single place and ease that package's replication. That's exactly what we did!

Now all the themes are located at app/Resources/themes/.

Setup

The configuration of which theme is active is divided in two files:

Which theme is active: domain.yml :

```yaml
multi_domain:
    hosts:
        upgradedirectory.arcasolutions.com:
            id: 20
            path: custom/domain_20/
            template: default
            locale: en-us
            localized: en
```

Which themes are available: config.yml :

```yaml
liip_theme:
    themes:
        - default
```