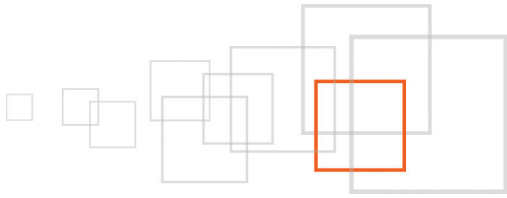
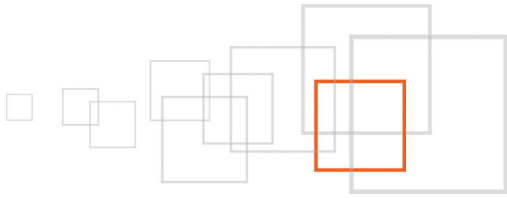
Adding custom security policy limitations to your modules

By Jérôme Vieilledent
http://www.lolart.net



Index



1 Goal description

This tutorial will show you how to deal with custom security policy limitations for your modules. Once read, you will be able to fully take advantage of the granularity in eZ Publish's security and access control system.

2 Introduction

As you might know, eZ Publish user access control is pretty precise and has a very fi ne granularity. Most of *kernel* modules allow indeed to limit access to themselves thanks to **security policies** we can assign to a user or to a user group. This is particularly the case for *content* module, fundamental within the CMS, thanks to its **function limitations** that can be configured in the admin interface. With these limitations, one can precisely define what a contributor has the right to do regarding content and functionalities.

3 Pre-requisites and target population

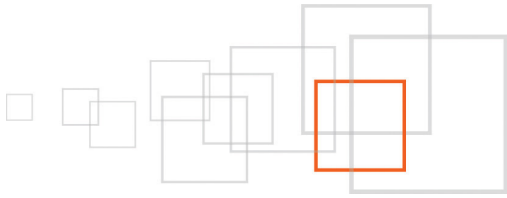
This tutorial is targeted to anyone who needs to implement security policy limitations for custom modules. Knowledge of eZ Publish module development is a must to fully understand the interest of this article. If you are not familiar to module and/or extension development ineZ Publish, you may read this excellent article about developing extensions. You also might read this interesting (old) tutorial about module development for eZ Publish(warning : this article was based on eZ Publish 3.x and PHP4, so read it with PHP5 style in mind). You can brush-up your knowledge of the basics of Access Control in eZ Publish by reviewing the online documentation on this subject.

4 Defining module functions to limit access to them

It is obviously possible to define security rules, configurable in the same way, for our own modules. All we need to do is to create a **list of functions** for our module, in **module.php**.

extension/myextension/modules/mymodule/module.php

```
<?php
$Module = array('name' => 'mymodule');


$ViewList = array();
$ViewList['myview'] = array(
      'script'                                => 'view.php',
      'params'                                => array(),
      'functions'                             => array( 'myfunction' )
);


$FunctionList['myfunction'] = array();
```



Here we define a *function* for the module **mymodule** and we affect it to the view **myview**. This function will display in the corresponding list when creating a new security policy for module **mymodule** in the admin interface.
You will find this kind of definition in most of modules contained in the kernel, or within extensions developped by eZ Systems or by the community (this is the case for NovenINIUpdate extension).

What I described above is sufficient in most cases, but what if we need to add more complex limitations like in content module (language, section, etc...) ?

5 Defining our own limitations

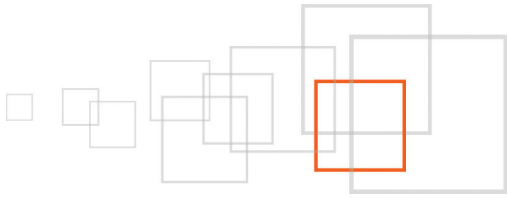
5.1 Definition

As you probably noticed in the *module.php* example above, variable **$FunctionList['myfunction']** is an empty array, which means that the function doesn't have any limitation. We just need to fi ll it with right values to add some.

Example for a language level limitation :

```
<?php
$Module = array('name' => 'mymodule');

$ViewList = array();
$ViewList['myview'] = array(
      'script'                                => 'view.php',
      'params'                                => array(),
      'functions'                             => array( 'myfunction' )
);

$Language = array(
    'name'=> 'Language',
    'values'=> array(),
    'path' => 'classes/',
    'file' => 'ezcontentlanguage.php',
    'class' => 'eZContentLanguage',
    'function' => 'fetchLimitationList',
    'parameter' => array( false )
);

$FunctionList['myfunction'] = array('Language' => $Language);
```

Here we tell eZ Publish that *myfunction* has a limitation named *Language* and that we need to fetch possible values list with *eZContentLanguage::fetchLimitationList()* method, with paramater *false*. This simply stands



for an old school way to define a callback function, probably here since first releases of eZ Publish 3 (remind you first versions of PHP4 and all its limitations). For curious minds, the important stuffs can be found in *eZPolicyLimitation::allValuesAsArrayWithNames()* , from line 249 (in a 4.3.0 eZ Publish instance).
Here we call a class from the kernel to fill our limitation array, but it is of course possible to use a class from an extension. However, autoload system cannot be used here and an old good *include_once* is made in the backend. So, in order to use a class from an extension, we need to add another key to our **$Language** array :

```
$Language = array(
    'name' => 'Language',
    'values' => array(),
    'extension' => 'myextension',
    'path' => 'classes/',
    'file' => 'myclass.php',
    'class' => 'MyClass',
    'function' => 'fetchLanguageLimitationList',
    'parameter' => array( false )
);
```

This way, *extension/myextension/classes/myclass.php* class will be included.

What does our method *MyClass:fetchLanguageLimitationList()* have to return ? A quick look to *eZContentLanguage::fetchLimitationList()* shows us that it must be a simple array with each entry being an associative array itself containing **id** and name **keys**.

- **id** will be the value stored in the database for the security policy limitation
- **name** will simply be the label displayed in the admin interface

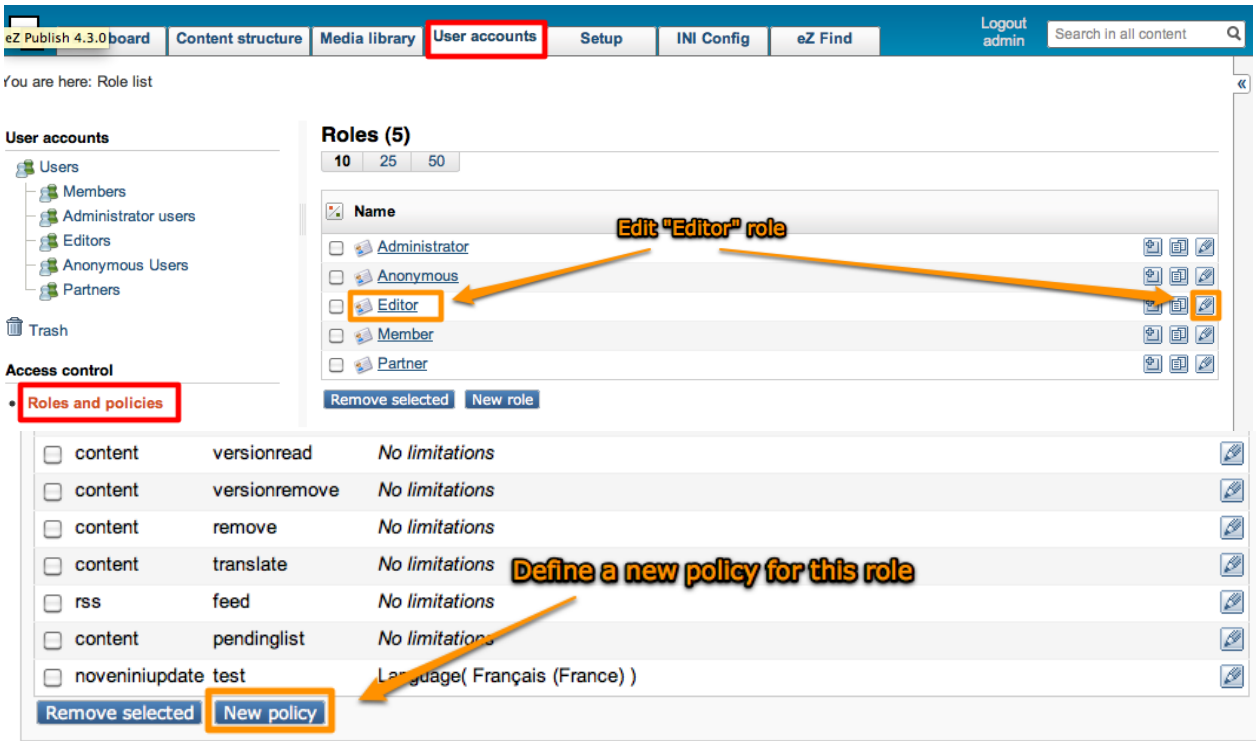
Here what should be our class *MyClass* :

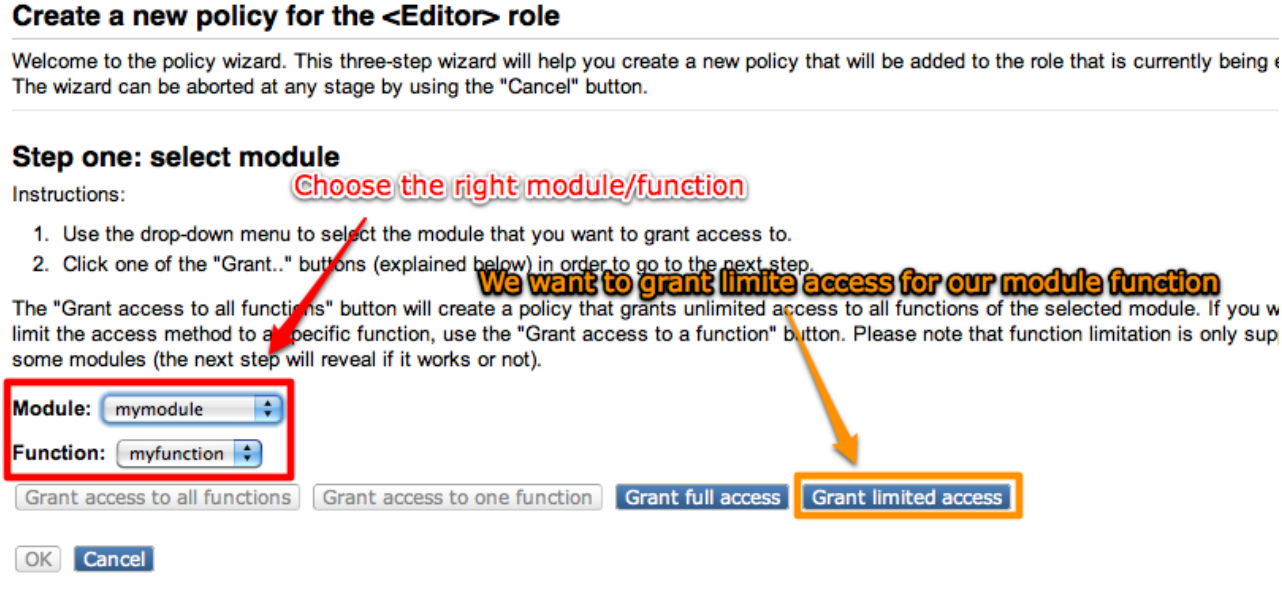
```
class MyClass
{
    /**
     * Fetches the array with names and IDs of the languages used on the site. This
method is used by the permission system.
     *
     * @return Array with names and IDs of the languages used on the site.
     * @static
     */
    public static function fetchLanguageLimitationList()
    {
      $langList = eZINI::instance( 'site.ini' )->variable( 'RegionalSettings',
'SiteLanguageList' );
      $aResult = array();
      foreach($langList as $lang)
      {
            if($lang)
```

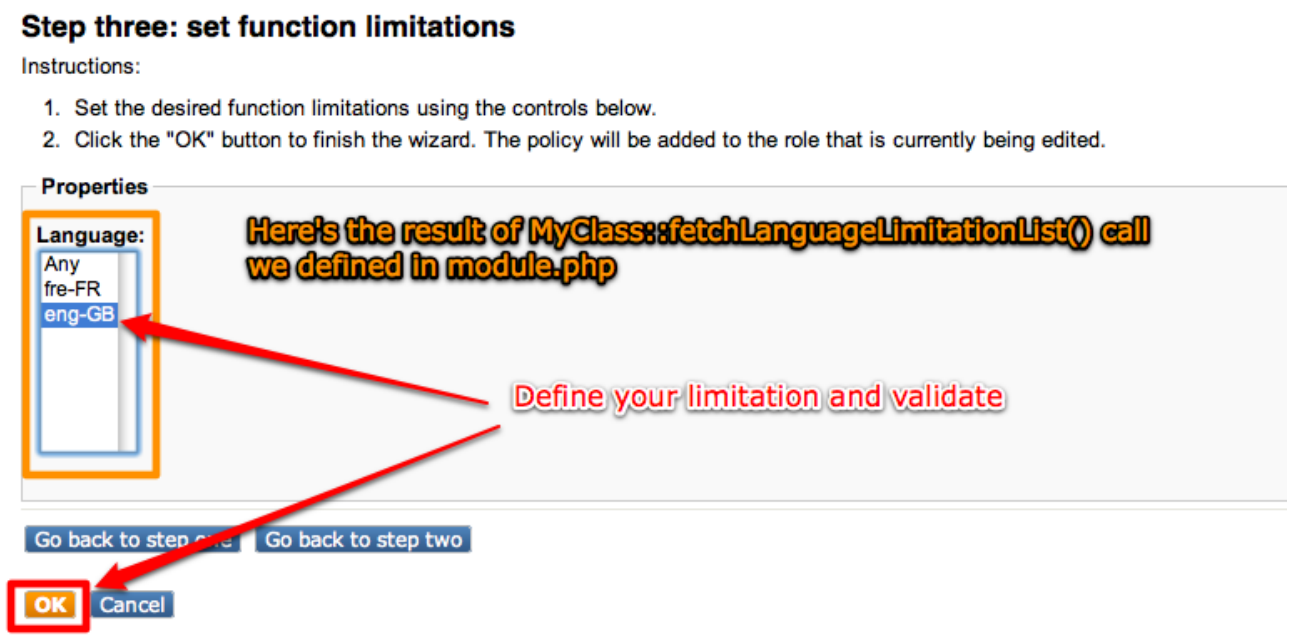
```
            {
                $aResult[] = array(
                    'id'   => $lang,
                    'name' => $lang
                );
            }
        }

        return $aResult;
    }
}
```

5.2 Adding the new policy to "Editor" role

Here is below a series of screenshots showing all the process :







6 Controlling access to our module

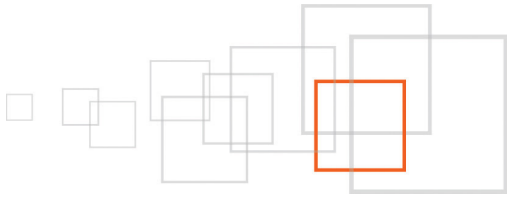
Now that our limitations are defined, **we now need to filter access to our module** depending on the rights that have been affected to a user. Indeed, this control is made in *index.php* for **content** module, and only for this one. As a consequence, we are forced to control the access ourselves. We can imagine changes in this regard in the future versions of eZ Publish.

6.1 Playing with eZJSCore

Manual access control of those limitations using the framework can easily become a real pain as the system is pretty complex. While a little refactoring would be nice on this part, workarounds exist that will make your life easier! Developers Łukasz Serwatka and Andrè Rømcke have implemented in eZJSCore extension a simplified access control method in the shape of a template operator. This extension being now part of eZ Publish distribution, it would be a shame not to use it ! Besides, you can find a real good article presenting this extension on the community portal.

Here is the best way to proceed :
extension/myextension/modules/mymodule/myview.php

```
$userHasAccess = ezjscAccessTemplateFunctions::hasAccessToLimitation( 'mymodule',
'myfunction' ); // Returns a boolean for current user
```



6.2 Get the limitations list for current user

Unfortunately, eZJSCore doesn't have (yet) any method allowing us to get available limitations for current user, which could be very useful to display a combo box containing limitations granted to the user for example (available languages in our case).
To do this, we will need to write a method returning those limitations in a simplifi ed way. Indeed, *eZUser* class does have *hasAccessTo()* method, but its result is absolutely unreadable and needs to be strongly simplifi ed. We will thus write a complementary method returning **simplified limitations**.

```
class MyClass
{
      /**
       * Shorthand method to check user access policy limitations for a given
module/policy function.
       * Returns the same array as eZUser::hasAccessTo(), with "simplifiedLimitations".
       * 'simplifiedLimitations' array holds all the limitations names as defined in
module.php.
       * If your limitation name is not defined as a key, then your user has full
access to this limitation
       * @param string $module Name of the module
       * @param string $function Name of the policy function ($FunctionList element in
module.php)
       * @return array
       */

      public static function getSimplifiedUserAccess( $module, $function )
      {
            $user = eZUser::currentUser();
            $userAccess = $user->hasAccessTo( $module, $function );

            $userAccess['simplifiedLimitations'] = array();
            if( $userAccess['accessWord'] == 'limited' )
            {
                  foreach( $userAccess['policies'] as $policy )
                  {
                        foreach( $policy as $limitationName => $limitationList )
                        {
                              foreach( $limitationList as $limitationValue )
                              {
                                    $userAccess['simplifiedLimitations']
[$limitationName][] = $limitationValue;
                              }
```

```
                                $userAccess['simplifiedLimitations'][$limitationName] =
array_unique($userAccess['simplifiedLimitations'][$limitationName]);
                            }
                        }
                }
                return $userAccess;
        }
}
```

This method returns an array containing result from *eZUser::hasAccessTo()*, with a new key : **simplifiedLimitations**. This key is also an array, containing the precious limitations.

In our example, for a user whom we would have affected a limitation *Language* allowing only *fre-FR* and *eng-GB*, this array would contain :

```
$limitations = MyClass::getSimplifiedUserAccess( 'mymodule', 'myfunction' );
print_r( $limitations['simplifiedLimitations'] );

// Result
Array
(
    [Language] => Array
        (
            [0] => fre-FR
            [1] => eng-GB
        )

)
```

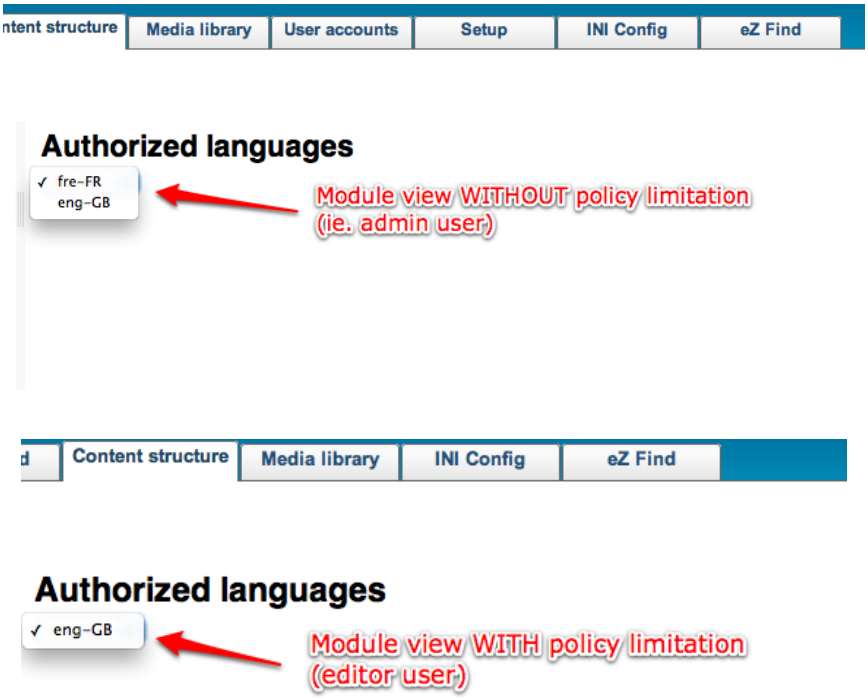
As a result, in our module where we need to display a combo box with authorized languages, we would have : *extension/myextension/modules/mymodule/myview.php*

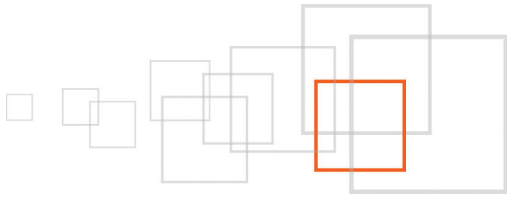
```
$tpl = eZTemplate::factory(); // Template init - from 4.3.0

$authorizedLang = eZINI::instance('site.ini')->variable( 'RegionalSettings',
'SiteLanguageList' ); // Default is all languages
$limitations = MyClass::getSimplifiedUserAccess( 'mymodule', 'myfunction' );
if( isset( $limitations['simplifiedLimitations']['Language'] ) ) // Found limitations
on language. These will be the only available in the dropdown menu
      $authorizedLang = $limitations['simplifiedLimitations']['Language'];
$tpl->setVariable( 'languages', $authorizedLang );

$Result['content'] = $tpl->fetch( 'design:mydesignsubdir/myview.tpl' );
```

extension/myextension/design/standard/templates/mydesignsubdir/myview.tpl

```
<select name="LanguageSelection">
{foreach $languages as $language}
      <option value="{$language}">{$language}</option>
{/foreach}
</select>
```

6.3 Result





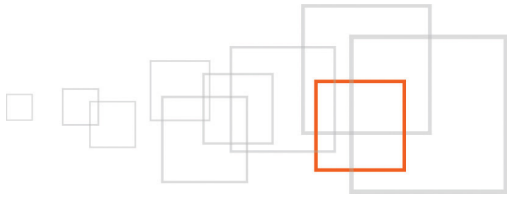
7 Security policies on tabs in the admin interface

eZ Publish 4.3.0 introduced the ability to filter the display of tabs in the admin interface depending on users security policies. Here is how to proceed :

extension/myextension/settings/menu.ini.append.php

```
<?php /* #?ini charset="utf-8"?

[NavigationPart]
Part[mynavigationpart]=My NavigationPart description

[TopAdminMenu]
Tabs[]=mytab

[Topmenu_mytab]
NavigationPartIdentifier= mynavigationpart
Name=INI Config
Tooltip=My Tooltip
URL[]
URL[default]=mymodule/myview
Enabled[]
Enabled[default]=true
Enabled[browse]=false
Enabled[edit]=false
Shown[]
Shown[default]=true
Shown[navigation]=true
Shown[browse]=false

# Simply add access control to your module's default function
PolicyList[]=mymodule/function

*/ ?>
```

Enjoy !



8 Conclusion

Here is a quick recap of each step :

1. Define a module *function* and affect it to one of the module *views*
2. Define a callback method in order to fetch available limitations for the *function*
3. Check user access in your module *view* with eZ JS Core
4. Fetch available limitations for current user with a helper static method
5. Control user access for tabs display in admin interface

As an example, we could consider NovenINIUpdate extension (environment switch for INI settings). It has indeed a GUI in the admin interface. At the time of writing, every user that has access to the main module of this extension can switch to every declared environment (ie. dev, staging, production). We could add a limitation on the environment, allowing many users to only switch from/to dev and staging environment. Production switch would be given only to admin users.

In short, building custom limitations for security policies attached to custom modules in eZ Publish is not a piece of cake. Personally, it took me several days of patience, deeply digging into the kernel to understand those mechanisms I just exposed to you, so you do not need to search for hours yourself :)
However, despite the real complexity , this mechanism allows very fi ne grain granularity in user access control. It is also unique and native in eZ Publish for a long time now.To be widely used, it would need to be simplifi ed within the kernel, as it has been done thanks to eZJSCore and the helper method we used in this tutorial.

I particularly would like to thank Nicolas Pastorino, Damien Pobel and Andrè Rømcke who helped me during my research :-).

9 Resources

This tutorial is an english translation of my original post on my blog, Lolart.net (french) :
http://www.lolart.net/blog/ez-publish/des-limitations-pour-vos-politiques-de-securite

9.1 Useful Articles and documentation :

- An introduction to developing eZ Publish extensions : http://share.ez.no/articles/ez-publish/an-introduction-to-developing-ez-publish-extensions
- Old tutorial about module development : http://ez.no/ezpublish/documentation/development/extensions/building_an_ez_publish_module
- eZ JS Core article : http://share.ez.no/articles/ez-publish/ezjscore-ez-publish-javascript-and-ajax-framework
- Noven INI Update : http://projects.ez.no/noveniniupdate
- Online documentation about Access Control management in eZ Publish : http://ez.no/doc/ez_publish/technical_manual/4_x/concepts_and_basics/access_control

10 About the author : Jérôme Vieilledent

Jérôme is a completely self-educated web developer. He learnt PHP all by himself and started developing with eZ Publish in 2007. He works now as a technical manager and expert at SQLi Agency in Paris.

11 License