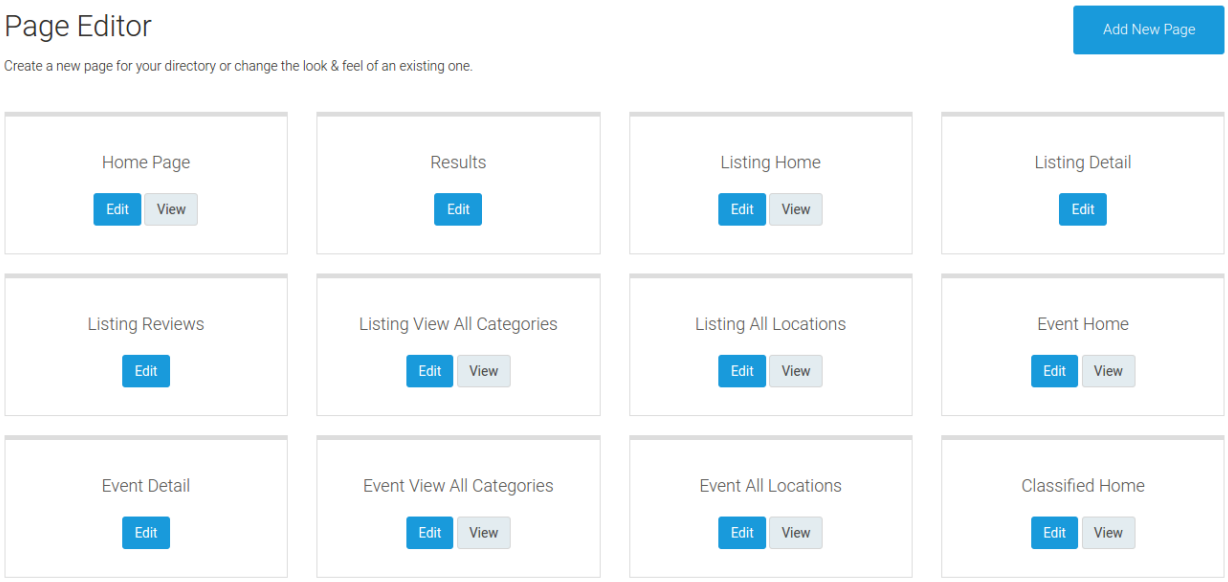
ARCA SOLUTIONS

eDirectory 11.2
Page Editor

Introdução

Page Editor é a nova ferramenta do eDirectory disponível para o site manager para controlar a aparência das páginas do frontend, criar novas páginas personalizadas, adicionar um conteúdo personalizado para uma página já existente ou editar as informações da página como título, URL e conteúdo de SEO.

Com o Page Editor você é capaz de ordenar os widgets de uma página como bem entender. Pode adicionar um widget a página, removê-lo ou editar seu conteúdo.

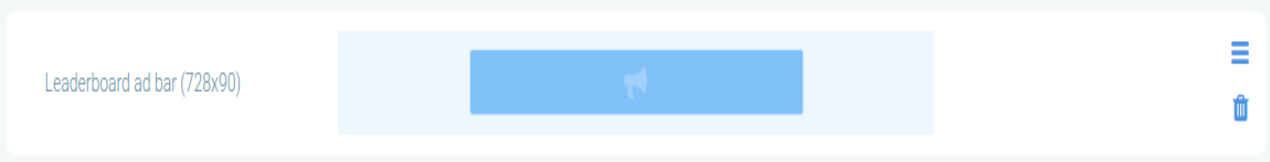


Página inicial do editor.

O que raios é um **WIDGET**?

Widget nada mais é que um bloco horizontal de conteúdo de uma página do frontend. Ainda está confuso? Vamos dar alguns exemplos:

- *Banner Leaderboard (728 x 90)*



Esta é a representação do widget de Banner Leaderboard no site manager.

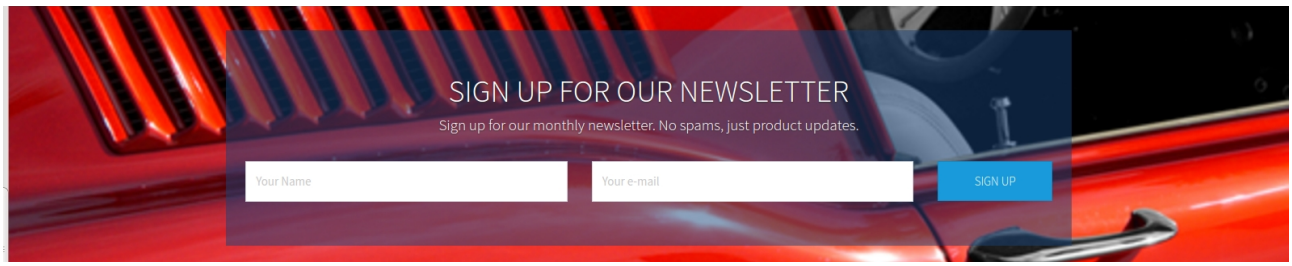
Ele consiste somente em um bloco com o um banner do tipo Leaderboard, é o exemplo mais simples de widget que temos disponível, não há nenhum conteúdo interno nele para o site manager customizar.

Aparência no frontend.

- *Signup for our newsletter*

Esta é a representação do widget de newsletter no site manager.

O widget de Newsletter já possui conteúdo que pode ser alterado pelo site manager, como o título, a descrição, e a imagem de fundo.
Repare que na lateral direita da imagem existe um ícone de edição( ), diferente do widget anterior.



Aparência no frontend.

Exitem outros widgets mais complexos como por exemplo o “Header”. Nele o site manager pode editar todos os labels da barra de login, bem como os itens da navegação do site, seus labels e links. E também pode alterar o logo.

* Nota Importante: Quando o site manager altera o conteúdo de algum widget do tipo “**Header**” ou “**Footer**” ele está alterando aquele widget para todas as páginas. Diferentemente dos outros widgets que seu conteúdo é diferente para cada página.

** Nota Importante – O império contra-ataca: Os widgets que só possuem campos de textos editáveis disponibilizam um checkbox para o caso de o site manager querer replicar as alterações feitas para aquele widget em todas as páginas. O widget “*Download our apps bar*” também se encaixa nesta regra.

Cada widget possui um arquivo *twig* que contém seu código *html*, e esses arquivos twig se encontram na pasta **widgets** de cada tema. Por exemplo no tema Restaurant o caminho é '`/Resources/themes/restaurant/widgets'`.

Todo widget pode ser realocado de posição pelo site manager. Basta arrastar o widget na posição que deseja e salvar. Por exemplo, podemos mover o widget de “*Header*” para abaixo do “*Footer*”.

O site manager pode adicionar ou remover qualquer widget de sua página. Salvo algumas exceções no quesito adicionar, pois alguns widgets só estão disponíveis para certas páginas. Por exemplo: o widget “*Result content with left filters*” só é disponível para a página “Results”. A lista completa de exceções pode ser encontrada no arquivo '`LoadWidgetPageTypeData.php'`.

Alguns grupos de widgets são limitados a somente um widget daquele grupo por página. Essa limitação se deve pelas possibilidades de conflitos de funcionalidades em javascript, ou simplesmente por não fazer sentido adicionar mais de um widget daquele na mesma página, por exemplo o “*Listing Detail*”.

```php
/**
 * This array contains the groups of the widgets that can only be one of them per page,
 * if widget's name changing at the load data is necessary change here as well.
 * You can add the title of the widget to block the whole group, or create another group.
 *
 * @var array
 */
private $widgetNonDuplicate = [
    'header'            => [
        'Header',
        'Header with Contact Phone',
        'Navigation with Centered Logo',
        'Navigation with left Logo plus Social Media',
    ],
```

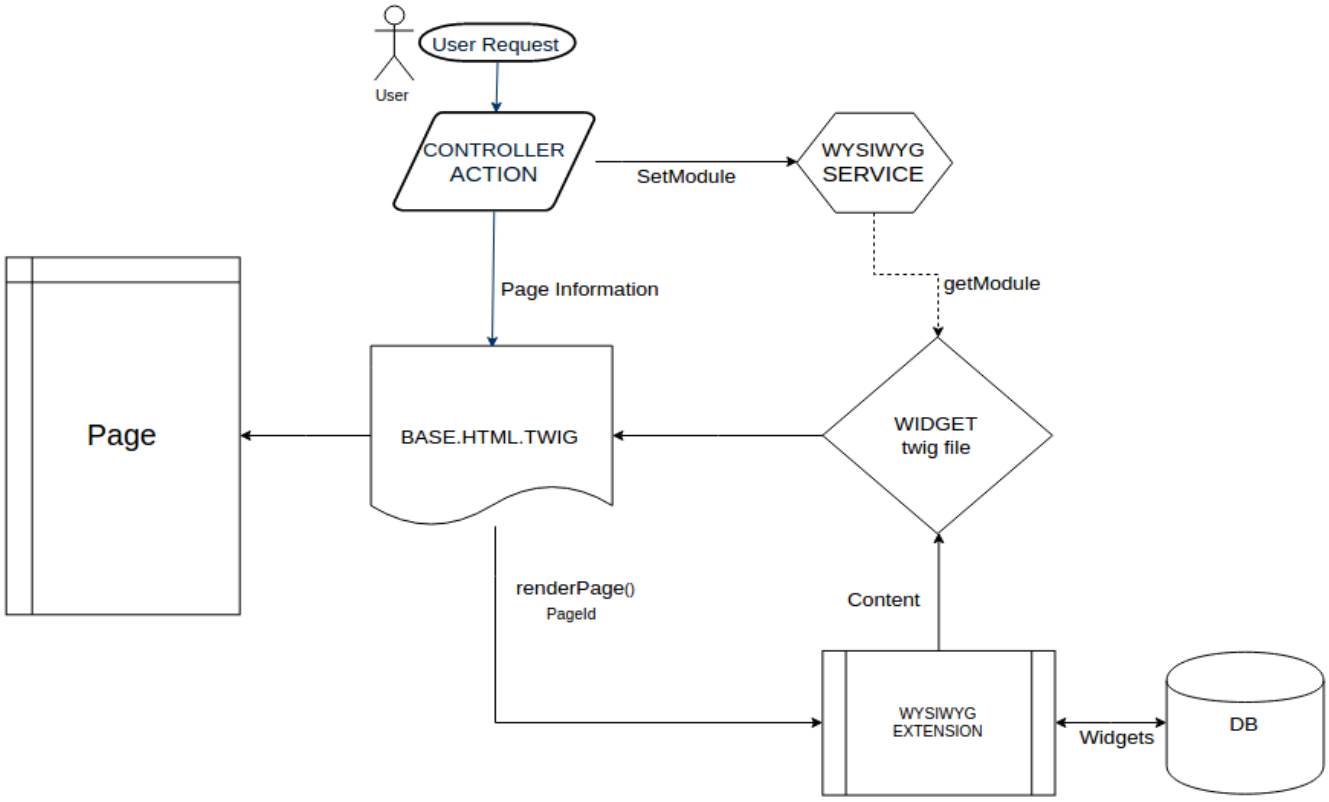
A lista completa de grupos de widgets que não podem ser duplicados se encontra na propriedade privada `$widgetNonDuplicate` no *Wysiwyg Service* (`Wysiwyg.php`).

Mal posso esperar para ver o que nossos clientes criativos vão fazer com essa funcionalidade do Page Editor.

A mágica por trás do Page Editor

Agora vamos falar de coisa boa, vocês conhecem a nova tekpix? Digo, vamos nos aprofundar na estrutura do Page Editor?

- *Frontend Workflow*



Com o advento do Page Editor o workflow do frontend sofreu algumas alterações. Agora quando o usuário faz alguma requisição ao sistema, ele cai na *Action* do *Controller* responsável pela rota requisitada e nela nós informamos ao *Wysiwyg Service* a qual módulo aquela *action* pertence, pois alguns widgets precisam saber a qual módulo eles pertencem para exibir seu conteúdo corretamente. Além de todos os widgets do tipo banner ou que contenham um banner, existem mais widgets que dependem do módulo, seguem **alguns exemplos**:

- `Browse by category block with images`
- `Reviews block`
- `All Locations`

Basicamente os widgets que tiverem conteúdo que possa variar conforme o módulo, como *categories*, *locations*, *reviews.*
Quando se trata de uma página que não pertence a nenhum módulo o *Wysiwyg Service* tem como padrão o módulo *listing*.

No final de toda *Action* nós retornamos o *Twig* daquela página e passamos para ela as informações da página (id,título, SEO). Com as mudanças do Page Editor, a *Twig* retornada será a `base.html.twig` na maioria dos casos.

* Nota Importante: Somente as *Actions* de *Results* e de *Detail* de módulo permanecem utilizando suas respectivas **twig** e não utilizam a `base.html.twig` como as demais.

Na `base.html.twig` é chamado o método *renderPage* da classe *WysiwygExtension*, passando o *id* da página. O método *renderPage* por sua vez recupera do banco de dados todos os widgets que pertencem aquela página ordenados, bem como o conteúdo que o site manager configurou para aquele widget.

O *renderPage* segue a ordem do campo **order** de cada widget configurada pelo site manager e vai montando a página final adicionando os arquivos *twig* do caminho contido no campo **twig_file** de cada widget e seu conteúdo (campo **content**).

* Nota Importante: O **content** que é passado para o arquivo *twig* é o da entidade *PageWidget*. O **content** da entidade *Widget* sempre será o padrão daquele widget, e nunca o configurado pelo site manager.

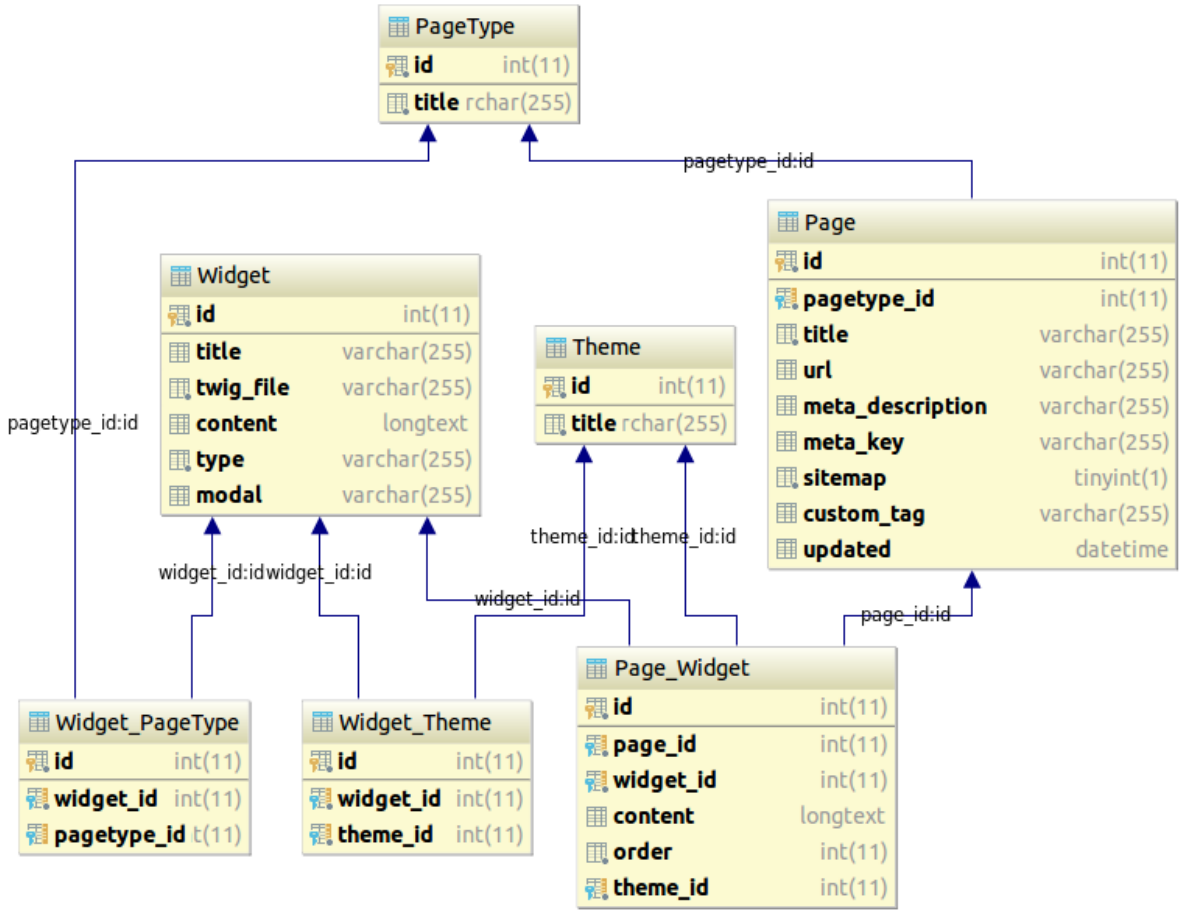
** Nota Important – Global offensive: As antigas váriáveis que eram enviadas pelo *Controller* para a twig de cada *Action* agora são variáveis globais das **twigs**, pois como agora para renderizar passamos por diversas **twigs** precisamos delas como referências globais. Tome cuidado para não utilizar variáveis com o mesmo nome. Exemplo de como adicionar uma variável global:

```
$twig = $this->container->get("twig");

$twig->addGlobal('item', $item);
```

Somente quando o *renderPage* terminou de montar todos os widgets é que retornamos a página para o usuário, marcando o fim do ciclo.

- O Banco de Dados



Uma parte importante de toda funcionalidade é entender o seu banco de dados, e com o Page Editor não seria diferente.

O primeiro ponto importante com relação ao banco de dados do Page Editor é entender que existem duas entidades principais, a entidade *Widget* e a *Page*. Toda a ideia da funcionalidade gira em torno destas duas entidades.

A tabela *Page* substituiu a antiga tabela *Content*, e teve seus campos com relação a título e SEO incorporadas pela *Page*. A coluna **type** tornou-se uma nova tabela, a *PageType*, que registra todos os tipos de páginas. Esses tipos estão listados no início do *Wysiwyg Service* como constantes.

Para o campo **content** da tabela *Content* foi criado um widget 'Custom Content' onde o site manager pode adicionar o conteúdo html que desejar. Portanto agora o site manager pode adicionar um código *html* em qualquer lugar da página através do widget 'Custom Content'.

A tabela *Widget* concentra as principais informações relacionadas aos widgets, entre elas: qual o arquivo twig (campo **twig_file**) que será renderizado quando o widget é

incluído em uma página, qual o conteúdo (**content**) editável pelo site manager, qual o *id* da modal (**modal***)* que permite a edição do widget para o site manager e qual o tipo (**type**) do widget que define em qual grupo ele será encaixado na modal de adicionar widgets.

Na tabela *Widget_Theme* consta qual widget está disponível para qual tema. Pois nem todos os widgets estão disponíveis para todos os temas.

A tabela *Widget_PageType* registra os widgets que são exclusivos para determinadas páginas. Ou seja, se um widget é exclusivo, haverá um registro nesta tabela entre o widget e o tipo de página. Caso contrário haverá somente um registro do widget sem página(null).

A tabela *Theme* contem os temas existentes do eDirectory.

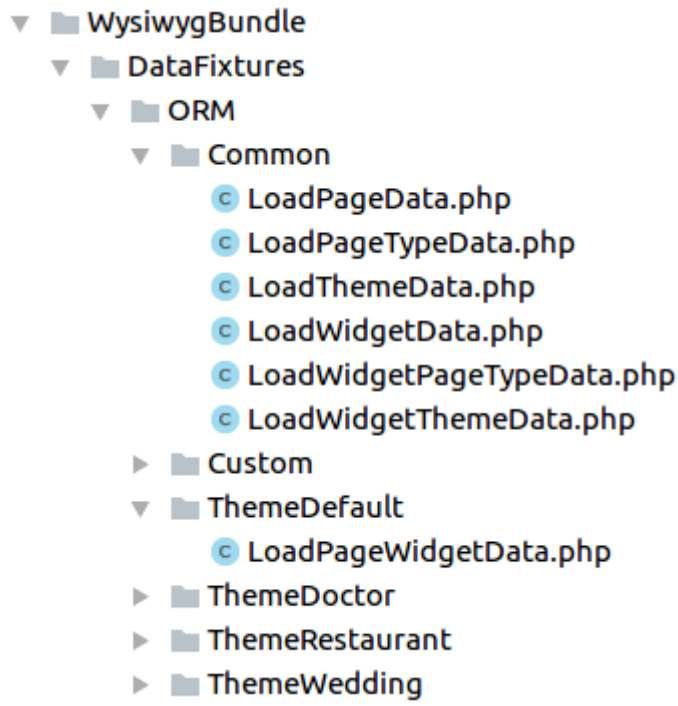
Enfim, a tabela *Page_Widget* faz o relacionamento entre as páginas e os widgets, ou seja, cada linha desta tabela contem um registro dizendo que uma página(**page_id**) possui um widget (**widget_id**) em uma determinada ordem (**order**), e também o conteúdo(**content**) configurado pelo site manager, para o tema(**theme_id**) indicado.

É nesta tabela que é possível descobrir quais os widgets que compõem a *Home Page* no tema *Default*, qual a ordem deles, e qual é o conteúdo de cada widget.

* Nota Importante: As imagens dos widgets não são gravadas na coluna **content**. Elas obedecem a estrutura da versão anterior. Como por exemplo o “slider” que possuia uma tabela específica para isso.

- O Load Data

Na versão 11.2 do eDirectory foi adicionado um novo bundle importante para o Page Editor chamado **DataFixtures**, que é responsável por fazer o carregamento padrão de todos os registros do banco de dados relacionado ao Page Editor.



Como quase todas as informações a respeito do Page Editor ficariam armazenadas no banco de dados, nós adicionamos este bundle para deixar registrado o conteúdo padrão das páginas do eDirectory.

O bundle transforma os objetos que você cria nele em inserções no banco de dados. Você define uma ordem de inserção das classes para obedecer as dependências do banco quanto a relacionamentos.

```php
/**
 * the order in which fixtures will be loaded
 * the lower the number, the sooner that this fixture is loaded
 *
 * @return int
 */
public function getOrder()
{
    return 2;
}
```

Para criar relacionamentos através do LoadData você precisa criar uma referencia para o objeto que deseja relacionar com outro, para que assim a próxima classe seguindo a ordem definida saiba encontrar o objeto para criar um relacionamento.

```php
$this->addReference($widget->getTitle(), $widget);
```

A documentação completa do bundle pode ser encontrada aqui.

- *Site manager*

O Page Editor está disponível no site manager na sessão Design & Customization. Nesta sessão o site manager pode gerenciar as páginas do eDirectory e seus widgets.

Alguns pontos importantes sobre a área do Page Editor:
- A funcionalidade em javascript da interface do Page Editor está concentrada nos arquivos '`web/scripts/widgets.js`' e '`sitemgr/assets/custom-js/widget.php`'.

- Todo wiget editável chama uma modal cujo *id* corresponde ao valor do campo **modal**. Os widgets que só possuem campos de texto para serem editados utilizam a mesma modal de *id* `'edit-generic-modal'`. E os que não possuem conteúdo editável possuem o campo **modal** vazio.

- Todo widget possui uma imagem de representação dele para cada tema. As imagens se encontram em: '`web/sitemgr/assets/img/widget-placeholder`'. É importante ressaltar que o nome de cada imagem se refere ao título do widget passando pela função '`system_generateFriendlyURL`', caso não haja uma imagem para aquele widget o sistema utiliza a imagem do "Custom Content" widget.

Outro ponto interessante é a funcionalidade de redefinir uma página, que permite ao site manager restaurar os widgets de uma página para seu formato padrão. Para isso, foi adicionado ao *Wysiwyg Service* (`Wysiwyg.php`) a configuração padrão das páginas do eDirectory. Para cada página existe uma função que retorna os widgets padrões daquela página para cada tema.

```php
/**
 * Returns the widgets that compose the Listing Home
 * Used for load data and reset feature at sitemgr
 *
 * @return mixed
 */
public function getListingHomeDefaultWidgets()
{
    $pageWidgetsTheme = [
        Theme::DEFAULT_THEME    => [
            'Header',
            'Search Bar',
            'Leaderboard ad bar (728x90)',
            '3 Featured Listings',
            'Browse by category block with images',
            'Best Of Listings',
            '3 rectangle ad bar',
            'Browse by Location with Right Banner (160x600)',
            'Banner Large Mobile, one banner Sponsored Links and one Google Ads',
            'Download our apps bar',
            'Footer',
        ],
```

Este é um trecho da função que retorna os widgets padrões da página "Listing Home".

A função completa de cada página pode ser encontrada no *Wysiwyg Service* (`Wysiwyg.php`).

```php
/**
 * Returns all the pages that have some widget that has any content different from its default
 * USED IN LOAD DATA
 *
 * @return array
 */
public function getDefaultSpecificWidgetContents()
{
    $translator = $this->container->get('translator');
    $language = substr($this->container->get("multi_domain.information")->getLocale(), 0, 2);

    // Set specific contents
    $contents = [];
    $contents[Wysiwyg::EVENT_HOME_PAGE]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Events']);
    $contents[Wysiwyg::CLASSIFIED_HOME_PAGE]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Classifieds']);
```

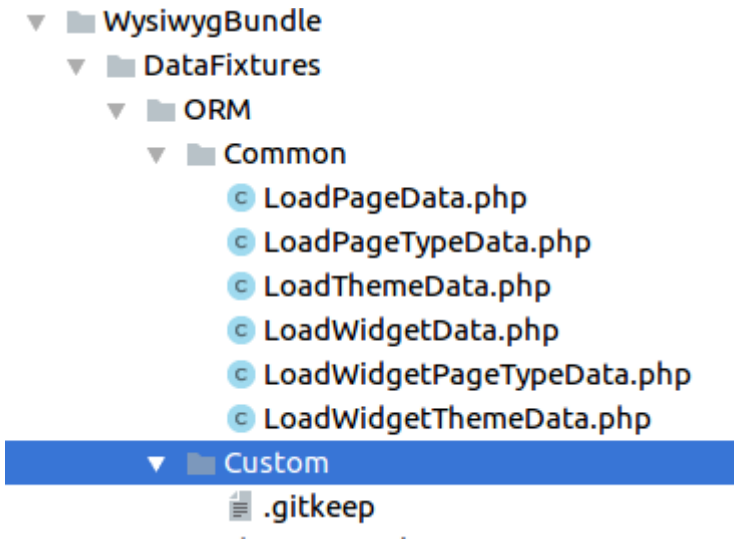
* Nota Importante: Algumas páginas possuem widgets com conteúdo diferente do padrão daquele widget. A lista completa se encontra na função '`getDefaultSpecificWidgetContents`' no *Wysiwyg Service* (`Wysiwyg.php`).

Customizando o Page Editor

Para customizar o Page Editor é preciso se atentar ao fato de que a maioria das informações ficam armazenadas no **banco de dados**, portanto é possível criar widgets somente a partir da inserção de suas informações no banco e a adição de sua *twig*, por exemplo. Porém esta não é a forma mais indicada e nem o foco deste guia.

As principais customizações envolvendo o Page Editor seriam a criação de novos widgets, páginas ou temas. A melhor forma de fazê-las é através do *LoadData*, assim tudo fica registrado no código e não somente no banco de dados.

Portanto para qualquer umas das customizações a seguir, adicionar um widget, página ou tema, é preciso copiar os arquivos da pasta **Common** do *LoadData* para dentro da pasta **Custom**.



* Nota Importante: Não esqueçam de corrigir o *namespace* dos arquivos copiados.

- *Adicionando um Widget*

Para este guia vamos usar de exemplo a criação do widget `'Popular Listings'`.

Para começar é preciso criar seu arquivo *twig* `'popular-listings.html.twig'`na sub pasta “listing” contida na pasta “widgets” de cada tema, por exemplo para o tema **default** seria em `'/Resources/themes/default/widgets/listing'`, a menos que o novo widget tenha exclusividade de temas, então só é preciso para os temas em que ele aparece.

Em seguida precisamos colocar as informações do widget no bundle de LoadData.

No recém copiado arquivo `'LoadWidgetData.php'`, no final do *array* `$standardWidgets` existe um comentário de exemplo de como adicionar um widget. Em caso de dúvidas, você deve criar um novo nó no *array* com as informações do

novo widget e seu conteúdo padrão. Exemplo:

```
[
    'title'    => 'Popular Listings',
    'twigFile' => '/listing/popular-listings.html.twig',
    'type'     => Widget::LISTING_TYPE,
    'content'  => [
        'labelPopularListings' => 'Popular Listings',
    ],
    'modal'    => 'edit-generic-modal',
],
```

Com isso, definimos o título, arquivo *twig*, o **type** para saber em qual aba ele será disponibilizado, o conteúdo com um único *label*, e como a única coisa editável dele serão *label(s)*, a **modal** genérica do novo widget `'Popular Listings'`.

* Nota Importante: Se o widget for mais complexo e o site manager puder editar mais coisas além de só *label(s)* é preciso criar uma modal específica para este novo widget na pasta das modais de widgets na área do site manager: `'/web/includes/modals/widget'`.

No recém copiado arquivo `'LoadWidgetPageTypeData.php'` iremos acrescentar, caso necessário, um nó no final do *array* `$exceptionsWidgets` com as páginas que este novo widget pode ser adicionado, se o novo widget for disponível para todas as páginas ignore esta etapa.

```
'Popular Listings'                      => [
    $this->getReference("TYPE_".Wysiwyg::HOME_PAGE),
    $this->getReference("TYPE_".Wysiwyg::LISTING_HOME_PAGE),
],
```

No nosso exemplo, definimos que o novo widget `'Popular Listings'`. Será exclusivo da `"Home Page"` e da `"Listing Home"`.

Agora para definir em quais temas o novo widget ficará disponível, é preciso adicionar um novo nó com o título do novo widget no *array* de retorno nas funções que definem quais os widgets de cada tema no *Wysiwyg Service* (`Wysiwyg.php`).

Cada tema tem a sua função listando seus widgets, e também existe uma para os widgets comuns a todos os temas.

```php
/**
 * Returns the commons and the Default Theme widgets
 *
 * @return array
 */
public function getDefaultThemeWidgets()
{
    $trans = $this->container->get('translator');

    return array_merge($this->getCommonThemeWidgets(), [
        $trans->trans('Header', [], 'widgets', 'en'),
        $trans->trans('Footer', [], 'widgets', 'en'),
        $trans->trans('Popular Listings', [], 'widgets', 'en'),
        /*
         * CUSTOM ADDWIDGET
         * here are an example of how you add the widget 'Widget test' for Default theme
         * if you need that 'Widget test' to be available for all themes you have
         * to remove it from here and add at the right function above
         *
         * $trans->trans('Widget test', [], 'widgets', 'en'),*/
    ]);
}
```

Neste caso deixamos nosso novo widget `'Popular Listings'` disponível somente no tema **default**. Caso precise ser comum a todos os temas era só adicionar no retorno da função '`getCommonThemeWidgets`'.

Pronto, agora só é preciso executar o comando no terminal para o *LoadData* inserir os novos dados, **não esqueça de colocar o domínio correto**.

```
php app/console doctrine:fixtures:load
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/Custom  --append
--domain=seu.dominio.com
```

As seguintes mensagens devem ser retornadas caso todos os passos tenham sido feitos corretamente:

```
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetData
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadThemeData
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadPageTypeData
> loading [2] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadPageData
> loading [2] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetThemeData
> loading [4] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetPageTypeData
```

Pronto, o novo widget `'Popular Listings'` foi adicionado.

- *Adicionando uma Página*

Para este guia vamos usar de exemplo a criação da página `"Categories Home"`, considerando que já foi adicionado uma rota, criado um *Controller* e uma *Action*, e tudo mais que era preciso no que diz respeito a adicionar uma nova página no front do eDirectory.

Adicionar uma nova página é bem semelhante a adicionar um widget. Assim como na adição de um novo widget, é preciso copiar os arquivos da pasta **Common** do *LoadData* para a pasta **Custom**.

* Nota Importante: Não esqueçam de corrigir o *namespace* dos arquivos copiados.

Antes de adicionar algo ao *LoadData* precisamos criar uma constante para o tipo desta nova página. Toda página tem o seu tipo listado como constante no *Wysiwyg Service* (`Wysiwyg.php`). Adicione o novo tipo no final da lista:

```
const BLOG_HOME_PAGE = "Blog Home";
const BLOG_DETAIL_PAGE = "Blog Detail";
const BLOG_CATEGORIES_PAGE = "Blog View All Categories";
/**
 * CUSTOM ADDPAGETYPE
 * here are an example of how you add a PageType constant to be used in the load data
 */
/*const TEST_PAGE = "Test Page";*/
const CATEGORIES_HOME = "Categories Home";
```

No recém copiado arquivo '`LoadPageTypeData.php`' adicione um novo tipo de página a um novo nó no final do *array* '`$standardPageTypes`'.

```
[
    'title' => Wysiwyg::ITEM_UNAVAILABLE_PAGE,
],
/*
 * CUSTOM ADDPAGETYPE
 * here are an example of how you add a PageType to be used in LoadPageData
 */
/* [
    'title' => Wysiwyg::TEST_PAGE,
],*/
[
    'tittle' => Wysiwyg::CATEGORIES_HOME,
]
```

No recém copiado arquivo '`LoadPageData.php`' adicione as informações da nova página a um novo nó no final do *array* '`$standardPages`'.

```
/**
 * CUSTOM ADDPAGE
 * Here are an example of how you add a page,
 * and you will need to create a PageType for this page
 *
 * Don't forget to create the alias for the url, if needed
 */
[
    'title'     => $trans->trans('Categories Home', [], 'widgets', 'en'),
    'url'       => $this->container->getParameter('alias_categories_home_url_divisor'),
    'metaDesc'  => '',
    'metaKey'   => '',
    'sitemap'   => false,
    'customTag' => '',
    'pageType'  => $this->getReference("TYPE_".Wysiwyg::CATEGORIES_HOME),
],
```

O próximo passo é criar a função que define quais os widgets padrões da página para cada tema. No *Wysiwyg Service* (`Wysiwyg.php`) crie uma função utilizando a constante com o novo tipo de página que foi adicionado como parte do nome da função, pois a funcionalidade de redefinir a página pega a função de acordo com o tipo da página.

```php
/**
 * Returns the widgets that compose the Categories Home
 * Used for load data and reset feature at sitemgr
 *
 * @return mixed
 */
public function getCategoriesHomeDefaultWidgets()
{
    $pageWidgetsTheme = [
        Theme::DEFAULT_THEME    => [
            'Header',
            'Search Bar',
            'Leaderboard ad bar (728x90)',
            'Browse by category block with images',
            '3 rectangle ad bar',
            'Banner Large Mobile, one banner Sponsored Links and one Google Ads',
            'Download our apps bar',
            'Footer',
        ],
        Theme::DOCTOR_THEME     => [...],
        Theme::RESTAURANT_THEME => [...],
        Theme::WEDDING_THEME    => [...],
    ];

    return $pageWidgetsTheme[$this->theme];
}
```

* Nota Importante: O nome desta função deve sempre obedecer a seguinte forma: '`get`' + '**valor da constante de tipo da página sem espaços**' + '`DefaultWidgets`'.

```php
/* Get Default Widgets Method */
$method = 'get'.str_replace(' ', '', $pageType).'DefaultWidgets';
$pageWidgetsArr = $this->$method();
```

** Nota Importante: Não esqueça de criar o *array* com os widgets da página para cada tema, e que cada tema tem seu widget de *Header* e *Footer*.

Uma vez criada a função que define o padrão dos widgets da página é preciso lista-lá para a função '`getAllPageDefaultWidgets`' utilizada no *LoadData*.

```php
/**
 * Returns an array with all the standard pages and its own array of default widgets
 * USED IN LOAD DATA
 *
 * @return array
 */
public function getAllPageDefaultWidgets()
{
    $pagesDefault = [];
    $pagesDefault[Wysiwyg::CATEGORIES_HOME] = $this->getHomePageDefaultWidgets();
    $pagesDefault[Wysiwyg::HOME_PAGE] = $this->getHomePageDefaultWidgets();
```

Se um dos widgets desta nova página possuir um conteúdo diferente somente para esta página é preciso definir seu conteúdo diferenciado na função '`getDefaultSpecificWidgetContents`'.

```php
/**
 * Returns all the pages that have some widget that has any content different from its default
 * USED IN LOAD DATA
 *
 * @return array
 */
public function getDefaultSpecificWidgetContents()
{
    $translator = $this->container->get('translator');
    $language = substr($this->container->get("multi_domain.information")->getLocale(), 0, 2);

    // Set specific contents
    $contents = [];
    $contents[Wysiwyg::CATEGORIES_HOME]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Categories']);
    $contents[Wysiwyg::EVENT_HOME_PAGE]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Events']);
```

Pronto, agora só é preciso executar o comando no terminal para o *LoadData* inserir os novos dados, **não esqueça de colocar o domínio correto**.

```
php app/console doctrine:fixtures:load
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/Custom  --append
--domain=seu.dominio.com
```

As seguintes mensagens devem ser retornadas caso todos os passos tenham sido feitos corretamente:

```
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetData
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadThemeData
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadPageTypeData
> loading [2] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadPageData
> loading [2] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetThemeData
> loading [4] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetPageTypeData
```

Pronto, a nova página `'Categories Home'` foi adicionada. Para finalizar, na edição da nova página dentro do Page Editor, clique em redefinir a página para os widgets padrão dela serem adicionados.

- *Adicionando um Tema*

Para este guia vamos usar de exemplo a criação do novo tema `'School'`, considerando que todo o processo prévio de criação de um novo tema foi seguido de acordo com este tutorial.

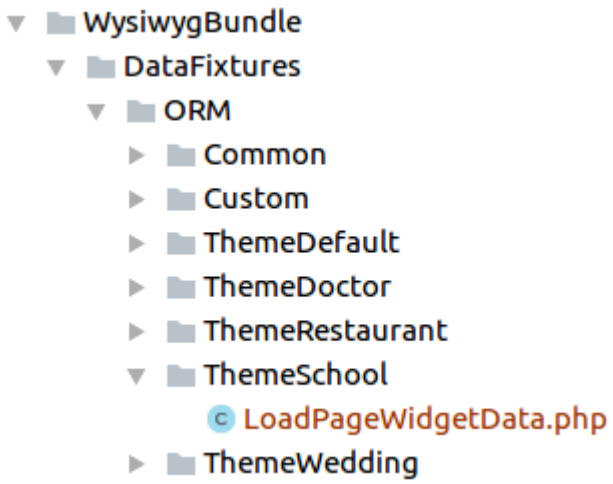
O primeiro passo é criar uma constante com o título do novo tema na entidade *Theme(*`Theme.php`*)*.

```php
/**
 * Existing themes titles of eDirectory
 */
const DEFAULT_THEME = 'Default';
const DOCTOR_THEME = 'Doctor';
const RESTAURANT_THEME = 'Restaurant';
const WEDDING_THEME = 'Wedding';
const SCHOOL_THEME = 'School';
```

No recém copiado '`LoadThemeData.php`' adicionar um novo nó no *array* '`$standardThemes`' com a constante do título do tema novo.

```php
/* Theme title is used as reference in LoadPageWidgetData,
 *  so if you change here don't forget to change there
 **/
$standardThemes = [
    Theme::DEFAULT_THEME,
    Theme::DOCTOR_THEME,
    Theme::RESTAURANT_THEME,
    Theme::WEDDING_THEME,
    Theme::SCHOOL_THEME,
    /**
     * CUSTOM ADDTHEME
     * here are an example of how you add the theme 'Test'
     *
     * Theme::TEST_THEME,  */
];
```

Diferente dos outros guias, para adicionar um Tema novo também é preciso criar uma nova pasta no *LoadData* para o novo tema. A pasta deve ser criada dentro da '**ORM**' com o nome de '**Theme**' mais o título do novo tema, no nosso exemplo fica '**ThemeSchool**'.



Dentro da nova pasta é preciso criar uma cópia do arquivo '`LoadPageWidgetData.php`' da pasta **ThemeDefault**. E nele alterar todas as chamas da constante do tema **Default**, para o nome do novo tema. **Exemplo**:

```php
$repository = $manager->getRepository('PageWidget');
$wysiwyg = $this->container->get('wysiwyg.service');
$wysiwyg->setTheme(Theme::SCHOOL_THEME);
$pagesDefault = $wysiwyg->getAllPageDefaultWidgets();
```

Agora falta fazer uma parte trabalhosa no *Wysiwyg Service (*`Wysiwyg.php`).

A primeira parte é simples, é preciso criar uma função que retorne um *array* com todos os widgets que estarão disponíveis neste novo tema. Por exemplo vamos dizer que nosso novo tema `'School'` é uma cópia do tema **Default**, portanto possuirá todos

os widgets deste tema em comum. A função ficaria assim:

```php
/**
 * Returns the commons and the School Theme widgets
 *
 * @return array
 */
public function getSchoolThemeWidgets()
{
    $trans = $this->container->get('translator');

    return array_merge($this->getCommonThemeWidgets(), [
        $trans->trans('Header', [], 'widgets', 'en'),
        $trans->trans('Footer', [], 'widgets', 'en'),
    ]);
}
```

O importante é a função retornar todos os widgets comuns a todos os temas mais os exclusivos deste novo tema. O nome da função ser formado por '`get`' + '**valor da constante do novo tema**' + '`ThemeWidgets`'. No nosso exemplo ficou: '`getSchoolThemeWidgets`'.

A segunda parte é mais trabalhosa, é preciso criar um novo nó no *array* de cada função que lista os widgets padrões de cada página do sistema. Este nó deve listar os widgets padrões da página para aquele novo tema. No exemplo abaixo efetuamos essa adição na função da "*Custom Page*".

```php
/**
 * Returns the widgets that compose the Custom Page
 * Used for load data and reset feature at sitemgr
 *
 * @return mixed
 */
public function getCustomPageDefaultWidgets()
{
    $pageWidgetsTheme = [
        Theme::DEFAULT_THEME    => [
            'Header',
            'Custom Content',
            'Footer',
        ],
        Theme::DOCTOR_THEME     => [...],
        Theme::RESTAURANT_THEME => [...],
        Theme::WEDDING_THEME    => [...],
        Theme::SCHOOL_THEME => [
            'Header',
            'Custom Content',
            'Footer',
        ]
    ];

    return $pageWidgetsTheme[$this->getSelectedTheme()->getTitle()];
}
```

Pronto, agora só é preciso executar o comando no terminal para o *LoadData* inserir os novos dados, **não esqueça de colocar o domínio correto**.

Note que existe **um parâmetro a mais onde está o caminho da pasta criada para este tema**, no nosso exemplo o tema `'School'` :

```
php app/console doctrine:fixtures:load
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/Custom
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/ThemeSchool
--append --domain=seu.dominio.com
```

As seguintes mensagens devem ser retornadas caso todos os passos tenham sido feitos corretamente:

```
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadThemeData
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetData
> loading [1] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadPageTypeData
> loading [2] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetThemeData
> loading [2] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadPageData
> loading [3] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\ThemeSchool\LoadPageWidgetData
> loading [4] ArcaSolutions\WysiwygBundle\DataFixtures\ORM\Custom\LoadWidgetPageTypeData
```

E pronto o novo tema `'School'` foi adicionado.