Newscoop Plugin Development Documentation

Release 4.2.1

SW

February 04, 2016

Contents

Plugins add extra functionality to Newscoop, an open source News CMS. This documentation shows you how to write plugins and contains simple examples you can use as a starting point.

Contents:

CHAPTER 1

Plugin Design

How to write your plugin for better integration with Newscoop.

You can also see Example Plugin code with all features described in this documentation.

1.1 Managing the Plugin Lifecycle

To manage the plugin from installation to removal, register the following event subscribers:

- plugin.install_vendor_plugin_name
- plugin.remove_vendor_plugin_name
- plugin update_vendor_plugin_name

`vendor_plugin_name` is the composer name property (vendor/plugin-name) with slashes / and hyphens - converted to underscores _.

This is an example of an event subscriber class containing placeholder functions for the three events:

```
// ExamplePluginBundle/EventListener/LifecycleSubscriber.php
<?php
namespace Newscoop\ExamplePluginBundle\EventListener;

use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Newscoop\EventDispatcher\Events\GenericEvent;

/**
 * Event lifecycle management
 */
class LifecycleSubscriber implements EventSubscriberInterface
{
    public function install(GenericEvent $event)
    {
        // do something on install
    }

    public function update(GenericEvent $event)
    {
        // do something on update
    }

    public function remove(GenericEvent $event)
    {
```

```
        // do something on remove
    }

    public static function getSubscribedEvents()
    {
        return array(
            'plugin.install.newscoop_example_plugin' => array('install', 1),
            'plugin.update.newscoop_example_plugin' => array('update', 1),
            'plugin.remove.newscoop_example_plugin' => array('remove', 1),
        );
    }
}
```

The next step is registering the class in the Event Dispatcher:

```
// ExamplePluginBundle/Resources/config/services.yml
services:
    newscoop_example_plugin.lifecyclesubscriber:
        class: Newscoop\ExamplePluginBundle\EventListener\LifecycleSubscriber
        tags:
            - { name: kernel.event_subscriber}
```

To provide access to all registered container services (`php application/console container:debug`), pass the services as the `@em` argument

```
// ExamplePluginBundle/Resources/config/services.yml
services:
    newscoop_example_plugin.lifecyclesubscriber:
        class: Newscoop\ExamplePluginBundle\EventListener\LifecycleSubscriber
        arguments:
            - @em
        tags:
            - { name: kernel.event_subscriber}
```

and use it in your service subscriber:

```
// ExamplePluginBundle/EventListener/LifecycleSubscriber.php
...
class LifecycleSubscriber implements EventSubscriberInterface
{
    private $em;

    public function __construct($em) {
        $this->em = $em;
    }
    ...
```

The Newscoop plugins system is based on the Symfony Bundles system, so almost all Symfony features are available. To create a new controller and route, start by creating the controller class:

```
<?php
// ExamplePluginBundle/Controller/LifecycleSubscriber.php

namespace Newscoop\ExamplePluginBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Symfony\Component\HttpFoundation\Request;
```

```
class DefaultController extends Controller
{
    /**
     * @Route("/testnewscoop")
     */
    public function indexAction(Request $request)
    {
        return $this->render('NewscoopExamplePluginBundle:Default:index.html.smarty');
    }
}
```

Note the annotation for route configuration `@Route("/testnewscoop")`. Register the controller class in the system:

```
// ExamplePluginBundle/Resources/config/routing.yml
newscoop_example_plugin:
    resource: "@NewscoopExamplePluginBundle/Controller/"
    type:     annotation
    prefix:   /
```

1.1.1 Working with views and templates

The previous Controller example returns a smarty template view:

```
return $this->render('NewscoopExamplePluginBundle:Default:index.html.smarty');
```

You can pass data from the controller to the view:

```
return $this->render('NewscoopExamplePluginBundle:Default:index.html.smarty', array(
    'variable' => 'super extra variable'
));
```

The original template is very simple:

```
// ExamplePluginBundle/Resources/views/Default/index.html.smarty
<h1>this is my variable {{ $variable }} !</h1>
```

For a more complex layout, use the Newscoop default publication theme layout `page.tpl`:

```
// ex. newscoop/themes/publication_1/theme_1/page.tpl
{{ include file="_tpl/_html-head.tpl" }}
<div id="wrapper">
    {{ include file="_tpl/header.tpl" }}
    <div id="content" class="clearfix">
        <section class="main entry page">
            {{ block content }}{{ /block }}
        </section>
        ...
    </div>
</div>
```

in the plugin template:

```
{{extends file="page.tpl"}}
{{block content}}
    <h1>this is my variable {{ $variable }} !</h1>
{{/block}}
```

1.2 Creating Database Entities

Newscoop uses Doctrine2 for database entity management:

- Get the entity manager from the Newscoop container using `$this->container->get('em');`
- Use the full FQN notation when getting entities: `$em->getRepository('Newscoop\ExamplePluginBundle\Entity`

1.3 Adding Admin Controllers

Admin Controllers consist of an action and a route, as in the example in `Newscoop\ExamplePluginBundle\Controller\DefaultController`. You can use Twig or Smarty as a template engine. There is information on extending the default admin layout, header, menu and footer in `Resources/views/Default/admin.html.twig`.

1.3.1 Adding a Plugin Menu to the Newscoop Admin Menu

The Newscoop Admin menu uses the KNP Menu Library and KNP MenuBundle. To add a Plugin Menu to the Admin Menu, add the service declaration:

```
newscoop_example_plugin.configure_menu_listener:
    class: Newscoop\ExamplePluginBundle\EventListener\ConfigureMenuListener
    arguments:
        - @translator
    tags:
      - { name: kernel.event_listener, event: newscoop_newscoop.menu_configure, method: onMenuConfigu
```

and the menu configuration listener to your plugin:

```
<?php
// EventListener/ConfigureMenuListener.php
namespace Newscoop\ExamplePluginBundle\EventListener;

use Newscoop\NewscoopBundle\Event\ConfigureMenuEvent;
use Symfony\Component\Translation\Translator;

class ConfigureMenuListener
{
    protected $translator;

    /**
     * @param Translator $translator
     */
    public function __construct(Translator $translator)
    {
        $this->translator = $translator;
    }

    public function onMenuConfigure(ConfigureMenuEvent $event)
    {
        $menu = $event->getMenu();
        $menu[$this->translator->trans('Plugins')]->addChild(
            'Example Plugin',
            array('uri' => $event->getRouter()->generate('newscoop_exampleplugin_default_admin'))
        );
```

```
    }
}
```

1.4 Adding Smarty Template Plugins

The Newscoop template language is Smarty3. Any Smarty3 plugins in

`<ExamplePluginBundle>/Resources/smartyPlugins`

are automatically loaded and available in your templates.

1.5 Adding Dashboard Widgets

The Newscoop admin panel automatically loads dashboard widgets from:

`<ExamplePluginBundle>/newscoopWidgets`

1.6 Plugin Hooks

Plugin hooks let you use existing Newscoop functionality in your plugins. Hooks are defined in PHP files in `<newscoopRoot>/admin-files/`:

- `issues/edit.php`
- `sections/edit.php`
- `articles/edit_html.php`
- `system_pref/index.php`
- `system_pref/do_edit.php`
- `pub/pub_form.php`

Example hook:

```
<?php
//newscoop/admin-files/articles/edit_html.php:

    echo \Zend_Registry::get('container')->getService('newscoop.plugins.service')
        ->renderPluginHooks('newscoop_admin.interface.article.edit.sidebar', null, array(
            'article' => $articleObj,
            'edit_mode' => $f_edit_mode
        ));
?>
```

1.6.1 Adding a Plugin Hook to your Plugin

Define the hook as a service, an addition to the article editing sidebar `articles/edit_html.php`:

```
//Resources/config/services.yml
newscoop_example_plugin.hooks.listener:
        class:     "Newscoop\ExamplePluginBundle\EventListener\HooksListener"
        arguments: ["@service_container"]
```

```
        tags:
          - { name: kernel.event_listener, event: newscoop_admin.interface.article.edit.sidebar, meth
```

In the `EventListener` folder of your plugin directory, `<ExamplePluginBundle>/EventListener` create `HooksListener.php` as specified in `services.yml` above:

```
<?php

namespace Newscoop\ExamplePluginBundle\EventListener;

use Symfony\Component\HttpFoundation\Request;
use Newscoop\EventDispatcher\Events\PluginHooksEvent;

class HooksListener
{
    private $container;

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

    public function sidebar(PluginHooksEvent $event)
    {
        $response = $this->container->get('templating')->renderResponse(
            'NewscoopExamplePluginBundle:Hooks:sidebar.html.twig',
            array(
                'pluginName' => 'ExamplePluginBundle',
                'info' => 'This is response from plugin hook!'
            )
        );

        $event->addHookResponse($response);
    }
}
```

The `sidebar()` method takes a `PluginHooksEvent` type as parameter. The PluginHooksEvent.php class collects `Response` objects from the plugin admin interface hooks.

Next, inside the `Resources/views` directory of your plugin create the `Hooks` directory we specified in the HooksListener. Then inside the `Hooks` directory create the view for the action: `sidebar.html.twig`.

```
<div class="articlebox" title="{{ pluginName }}">
    <p>{{ info }}</p>
</div>
```

The plugin response from the hook shows up in the article editing view:

1.7 User Permissions in Plugins

A guide to restricting access to resources in your plugin to certain users.

Add a *PermissionsListener* class where you define plugin permissions:

```
<?php
namespace Acme\DemoPluginBundle\EventListener;
```

```
use Newscoop\EventDispatcher\Events\PluginPermissionsEvent;
use Symfony\Component\Translation\Translator;

class PermissionsListener
{
    /**
     * Translator
     * @var Translator
     */
     protected $translator;

     public function __construct(Translator $translator)
     {
         $this->translator = $translator;
     }

     /**
      * Register plugin permissions in Newscoop ACL
      *
      * @param PluginPermissionsEvent $event
      */
     public function registerPermissions(PluginPermissionsEvent $event)
     {
         $event->registerPermissions($this->translator->trans('ads.menu.name'), array(
             'plugin_classifieds_edit' => $this->translator->trans('ads.permissions.edit'),
         ));
     }
 }
```

The first parameter of the *registerPermissions()* method is a custom plugin name, the second parameter is an array of permissions where each key is a unique permission name and each value is a translated permission label.

For example, *plugin_classifieds_edit* is the unique permission name, and the translated permission label should be in the following form:

```
plugin.plugin_name.permission_name
```

Where:

- *plugin* - plugin namspace, for example *ads*
- *plugin_name* - plugin name, for example *permissions*
- *permission_name* - permission name, add, delete, etc, for example *edit*

Registering the listener in *services.yml*:

```
#Acme\DemoPluginBundle\Resources\config\services.yml
services:
    acme_demo_plugin.permissions.listener:
        class: Acme\DemoPluginBundle\EventListener\PermissionsListener
        arguments:
            - @translator
        tags:
          - { name: kernel.event_listener, event: newscoop.plugins.permissions.register, method: regi
```

To check if a user has been given a permission, call **hasPermission()** method on *User* object:

```
$user->hasPermission('plugin_classifieds_edit');
```

1.7.1 Registering Permissions on Plugin Install/update

To register permissions during plugin installation or update process, create a method in *LifecycleSubscriber.php*:

```
<?php
//Acme\DemoPluginBundle\EventListener\LifecycleSubscriber.php

/**
 * Collect plugin permissions
 */
private function setPermissions()
{
    $this->pluginsService->savePluginPermissions($this->pluginsService->collectPermissions($this->tra
}
```

Then during plugin installation, call the *setPermissions()* method that you created:

```
<?php
//Acme\DemoPluginBundle\EventListener\LifecycleSubscriber.php

public function install(GenericEvent $event)
{
    $tool = new \Doctrine\ORM\Tools\SchemaTool($this->em);
    $tool->updateSchema($this->getClasses(), true);

    $this->em->getProxyFactory()->generateProxyClasses($this->getClasses(), __DIR__ . '/../../../../I
    $this->setPermissions();
}
```

1.7.2 Checking Permissions in Views - Twig Extension

Check user permissions in Twig templates:

```
{% if hasPermission('plugin_classifieds_delete') %}
   <!-- user has delete permission, do some stuff here -->
{% endif %}
```

1.8 Plugin Cron Jobs

Newscoop 4.3 introduces a new cron job management system, which also affects repetitive tasks in plugins.

Before Newscoop 4.3, to call a function ever few hours you would create a *Console Command* in the *AcmeExamplePluginBundleCommand* namespace.

In Newscoop 4.3 you now use *TestCronJobCommand*. The following example prints *Test cron job command.*.

```
<?php

namespace Acme\ExamplePluginBundle\Command;

use Symfony\Component\Console;
use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;

/**
 * Test cron job command
 */
```

```
class TestCronJobCommand extends ContainerAwareCommand
{
    /**
     */
    protected function configure()
    {
        $this->setName('example:test')
            ->setDescription('Example test cron job command');
    }

    /**
     */
    protected function execute(Console\Input\InputInterface $input, Console\Output\OutputInterface $o
    {
        try {
            $output->writeln('<info>Test cron job command.</info>');
        } catch (\Exception $e) {
            $output->writeln('<error>Error occured: '.$e->getMessage().'</error>');

            return false;
        }
    }
}
```

To run the cron job, register it on plugin install and update.

1.8.1 Registering Cron Jobs on Plugin Install/Update

To register a cron job during the plugin install/update process, edit the *LifecycleSubscriber.php* class.

Add the *newscoop.scheduler* service to *LifecycleSubscriber* class.

```
services:
    newscoop_example_plugin.lifecyclesubscriber:
        class: Newscoop\ExamplePluginBundle\EventListener\LifecycleSubscriber
        arguments:
            - @em
            - @newscoop.scheduler
```

Add a new property called *cronjobs* which is an array of our plugin cron jobs.

```
<?php
//Acme\ExamplePluginBundle\EventListener\LifecycleSubscriber.php

protected $scheduler;

protected $cronjobs;

public function __construct(EntityManager $em, SchedulerService $scheduler)
{
    $appDirectory = realpath(__DIR__.'/../../../../application/console');
    $this->em = $em;
    $this->scheduler = $scheduler;
    $this->cronjobs = array(
        "Example plugin test cron job" => array(
            'command' => $appDirectory . ' example:test',
            'schedule' => '* * * * *',
        ),
```

```
        /*"Another test cron job" => array(
            'command' => $appDirectory . ' example:anothertest',
            'schedule' => '* * * * *',
        ),*/
    );
}
```

Use any of the following parameters to define cron jobs:

- string *command* (**required**) The job to run, either a shell command or an anonymous PHP function. In this example it's our *TestCronJobCommand*
- string *schedule* (**required**) Crontab schedule format (*man -s 5 crontab*)
- boolean *enabled* Run this job at scheduled times
- boolean *debug* Send *scheduler* internal messages to 'debug.log'
- string *dateFormat* Format for dates on scheduler log messages
- string *output* Redirect *stdout* and *stderr* to this file
- string *runOnHost* Run jobs only on this hostname
- string *environment* Development environment for this job
- string *runAs* Run as this user, if crontab user has *sudo* privileges

Create a method in the same class to add the cron jobs.

```
<?php
//Acme\ExamplePluginBundle\EventListener\LifecycleSubscriber.php

/**
 * Add plugin cron jobs
 */
private function addJobs()
{
    foreach ($this->cronjobs as $jobName => $jobConfig) {
        $this->scheduler->registerJob($jobName, $jobConfig);
    }
}
```

And add the *addJobs* methor to the install/update event:

```
<?php
//Acme\DemoPluginBundle\EventListener\LifecycleSubscriber.php

public function install(GenericEvent $event)
{
    $tool = new \Doctrine\ORM\Tools\SchemaTool($this->em);
    $tool->updateSchema($this->getClasses(), true);

    $this->em->getProxyFactory()->generateProxyClasses($this->getClasses(), __DIR__ . '/../../../../
    $this->addJobs();
}
```

Now, when you install the plugin, the *Example plugin test cron job* is inserted into the database, and can be managed via *System Preferences -> Background Jobs Settings*. Plugin 8 in the example screenshot:

1.8.2 Removing Registered Cron Jobs on Plugin Remove Event

Cron jobs that are installed by a plugin also need to be removed when the plugin is uninstalled.

Add a function to remove the cron job:

```
<?php
//Acme\ExamplePluginBundle\EventListener\LifecycleSubscriber.php

/**
 * Remove plugin cron jobs
 */
private function removeJobs()
{
    foreach ($this->cronjobs as $jobName => $jobConfig) {
        $this->scheduler->removeJob($jobName, $jobConfig);
    }
}
```

and call it during plugin *remove* event:

```
<?php
//Acme\ExamplePluginBundle\EventListener\LifecycleSubscriber.php
public function remove(GenericEvent $event)
{
    $tool = new \Doctrine\ORM\Tools\SchemaTool($this->em);
    $tool->dropSchema($this->getClasses(), true);
    $this->removeJobs();
}
```

- search

Find a complete list of available plugins in the Official Plugin Repository.