How to Upgrade to Symfony 2.8, then 3.0!

With <3 from SymfonyCasts

Chapter 1: How to Upgrade to Symfony 2.8

Confused by the First Paragraph? https://www.youtube.com/watch?v=K7zNY0l5JNI

The Symfony Demo: example app: a project barely alive... on Symfony 2.7. Ladies and gentlemen, we can rebuild it, we have the technology. We have the capability to make the world's first Symfony 3 app. Better than it was before, stronger, faster, the 3.0 version demo app. This series is all about upgrading to Symfony 3.

Of course, the first step is actually to upgrade to Symfony 2.8. No wait! That's really cool too: because it'll give us all of the features of Symfony 3.

To get to 2.8, open the composer.json file, find the symfony/symfony line and set it to 2.8. At recording, Symfony 2.8 wasn't released yet so I'll set this to 2.8.*@BETA:

```
62 lines | composer.json
... lines 1 - 8
9       "require": {
... lines 10 - 26
27          "symfony/symfony"                   : "2.8.*@BETA",
... line 28
29      },
... lines 30 - 62
```

But you - person-living-in-the-future - you should set it to 2.8.*.

Head over to the terminal and run:

```
$ composer update symfony/symfony --with-dependencies
```

This will download 2.8 *and* update any other libraries that Symfony depends on.

Yes yes, I realize that we're ignoring potential updates of all of these other bundles and libraries. But focus! Upgrade Symfony first, and *then* upgrade other libraries when you want to.

Changes in 2.8 (Deprecations)

Since we're just upgrading from Symfony 2.7 to 2.8, there *shouldn't* be any backwards compatibility breaks... though one usually sneaks in... usually with forms. While Composer is dialing out to Jordi, switch to the browser, and head to http://github.com/symfony/symfony to check out the upgrade log for 2.8 to see what *has* changed.

This describes things that were deprecated. Translation: these are things you'll need to upgrade before heading to 3.0... but you can safely ignore them for now. I've already covered some of these in our tutorial on What's new in Symfony 3. For example, the new way you create form types has changed quite a lot. But this isn't our concern quite yet.

Head back to the terminal to check on our update. Cool it's done! Scroll up to see that it updated to Symfony 2.8 Beta 1, and grabbed a whole bunch of other library updates related to this.

Go over to the browser and refresh our localhost homepage. That's it! We've now got the new web debug toolbar at the bottom: ooo pretty. That's your sign that you are now upgraded to 2.8!

That is the easiest step in this process... and it's also the most important because Symfony 2.8 has all the features of 3.0. So yea, woo! Next we'll upgrade to the new, fancy, Symfony 3 directory structure.

Chapter 2: Upgrading to the Symfony 3.0 Directory Structure

Hey, we're on Symfony 2.8! Before we *fully* upgrade, let's move into the new 3.0 directory structure... which you can actually use with any version of Symfony... But in 3.0, it's super official. And remember: the new structure isn't required. Want to keep things how they are now? Do it! We talk more about the new structure in our What's new in Symfony 3 screencast.

People often ask me:

> What are the first 10 digits of pi?

Great question: they're 3.1415926535. They also ask me:

> What do I need to change in my code after upgrading?

The answer to that is... nothing! Ya know, because Symfony doesn't break backwards compatibility.

But we *do* sometimes tweak things in the default directory structure of Symfony. A sure-fire way to find out about those changes is to go to the Symfony Standard repository and just compare the versions. If you compare 2.8 to master, you'll see all the changes needed to move into this new directory structure.

Ready to do this?

Autoload AppKernel

In each front controller we require the AppKernel file:

```
39 lines | web/app_dev.php
... lines 1 - 30
31 require_once __DIR__.'/../app/AppKernel.php';
... lines 32 - 39
```

That's because this file doesn't have a namespace and doesn't live in src. Basically, it's weird, and can't be autoloaded.

That was lame, so it *is* autoloaded now! In composer.json, add a files key with app/AppKernel.php inside of it:

```
63 lines | composer.json
... lines 1 - 5
6     "autoload": {
7         "psr-4": { "": "src/" },
8         "files": ["app/AppKernel.php"]
9     },
... lines 10 - 63
```

With that, we can rip out the require statement in app.php, app_dev.php, app/console and in AppCache.php. Nice!

For this to take effect, head over to the terminal and dump the autoloader:

```
$ composer dump-autoload
```

Running composer install also does this.

Refresh! And it still works!

The var/ Directory

The most noticeable change is the new var/ directory which holds stuff you don't need to modify, like the cache and logs. To update your project, open AppKernel and override a few methods. I'll use command+n to open the generate menu. Select override and choose getRootDir(), getCacheDir(), and getLogDir(). I'm not sure why we override getRootDir(), because that

defaults to *this* directory, but I *do* like that it looks less magic now:

63 lines | app/AppKernel.php

```
... lines 1 - 5
class AppKernel extends Kernel
{
... lines 8 - 47
    public function getRootDir()
    {
        return __DIR__;
    }
... lines 52 - 61
}
```

For getCacheDir(), return dirName(__DIR__) - that looks in app/ then goes up to the root - and then add .'/var/cache/'.$this->environment;:

63 lines | app/AppKernel.php

```
... lines 1 - 52
    public function getCacheDir()
    {
        return dirname(__DIR__).'/var/cache/'.$this->environment;
    }
... lines 57 - 63
```

Do a similar thing in getLogDir() except change to .'/var/logs/';:

63 lines | app/AppKernel.php

```
... lines 1 - 57
    public function getLogDir()
    {
        return dirname(__DIR__).'/var/logs';
    }
... lines 62 - 63
```

That's *all* you need to do to move the cache & logs directories.

In the terminal create a var/ directory, and then git mv app/cache/ into var/. Do the same to move app/logs/ into var/:

```
$ mkdir var
$ git mv app/cache var/
$ git mv app/logs var/
```

This might seem weird to you because the cache and logs directories aren't normally stored in git. But we *do* keep a .gitkeep file inside each to make sure they don't get deleted. That's what we just moved.

Back to the browser and refresh! It's still alive! In our IDE, we have a new var/ directory that's happily populated with cache and logs files.

Changes to bin/

Next, look at the bin directory. It exists because composer.json holds a little bit of configuration: "config:" "bin-dir:":

63 lines | composer.json

```
{
... lines 2 - 51
    "config": {
        "bin-dir": "bin"
    },
... lines 55 - 61
}
```

Sometimes when you install an external library via composer, it comes with a binary file. Composer usually puts that into a vendor/bin directory for convenience. With this config, it goes into bin/ instead?

Why did we override this? I have no idea! Composer was young, it was the wild-west, things happen. And really, it's not a big deal, except that it makes Symfony projects look a bit "weird" compared to others. Remove this config so that Composer uses the normal vendor/bin.

In the terminal, completely remove the bin/ directory. But wait! The bin directory *does* still exist in the Symfony 3 directory structure, it just has a new job. Put it back!

```
$ rm -rf bin
$ mkdir bin
```

bin/console

Phew! Now that it's back, move the app/console file into bin/:

```
$ mv app/console bin/console
```

This is one of the biggest changes. Goodbye app/console, hello bin/console. Of course this file is angry because the bootstrap.php.cache file is not in the same directory anymore. But instead of loading it, load /../app/autoload.php instead:

27 lines | bin/console

```
... lines 1 - 9
require __DIR__.'/../app/autoload.php';
... lines 11 - 27
```

Why? Well, in part because bootstrap.php.cache is going to move too.

Head to the terminal and try out your very first bin/console:

```
$ bin/console
```

It's still happy, and I'm still happy.

Moving bootstrap.php.cache

Speaking of bootstrap.php.cache, this now lives in the var directory. But wait, *we* aren't responsible for this file. In fact, who created this in the first place? Who put this in the app directory to begin with? It was Santa! Just kidding: it's the SensioDistributionBundle, via one of these post install hooks in composer.json:

60 lines | composer.json

```
... lines 1 - 33
34      "scripts": {
35          "post-install-cmd": [
... line 36
37              "Sensio\\Bundle\\DistributionBundle\\Composer\\ScriptHandler::buildBootstrap",
... lines 38 - 41
42          ],
... lines 43 - 50
51      },
... lines 52 - 60
```

We need to tell the SensioDistributionBundle that we want it to be built in var instead of app. To do that, add a couple of new configuration lines in the extra section of composer.json.

Add "symfony-var-dir": set to "var", "symfony-bin-dir": set to "bin" and "symfony-tests-dir": set to "tests". The really important one here symfony-var-dir:

63 lines | composer.json

```
... lines 1 - 51
52      "extra": {
53          "symfony-app-dir": "app",
54          "symfony-web-dir": "web",
55          "symfony-var-dir": "var",
56          "symfony-bin-dir": "bin",
57          "symfony-tests-dir": "tests",
... lines 58 - 60
61      }
... lines 62 - 63
```

When you run composer install it's going to see this and realize that you want it to put the bootstrap.php.cache file into the var directory. It's as simple as that.

The symfony-bin-dir is *also* important because of the check.php and SymfonyRequirements.php files. These also moved, and the SensioDistributionBundle is *also* in charge of putting those inside of either the app, bin or var directories. In a second, you'll see these move.

The symfony-test-dir, well, as far as I can see, that doesn't do anything. But it exists in the official Symfony Standard Edition and darn it, I'm a rule-follower.

Back to our favorite place, the terminal! Run:

```
$ composer install
```

Nothing is installing, but it does run the post install scripts. And... ding! Suddenly a few files disappeared from the app directory: check.php and SymfonyRequirements.php. check.php is now called bin/symfony_requirements and the SymfonyRequirements file has been moved over to the var directory. All of that was done thanks to that extra configuration.

Require autoload.php instead of bootstrap.php.cache

In the var directory, there's the bootstrap.php.cache file. Delete the original one in app. To un-break our application, we need to go into app_dev.php and require the new path to the bootstrap file. But stop! Like the console file, we're no longer going to include bootstrap.php.cache. Instead require app/autoload.php:

37 lines | web/app_dev.php

```
... lines 1 - 27
28  $loader = require __DIR__.'/../app/autoload.php';
... lines 29 - 37
```

Tip

Actually, the require_once was also changed to require, which guarantees that the $loader is returned.

Why? Well, the *whole* point of bootstrap.php.cache is performance: by having a bunch of classes in one file, it means that the autoloading mechanism has to work less. But when you're debugging, it's not helpful to see errors coming from a deep line in bootstrap.php.cache.

Instead, in the dev environment, only include the normal autoloader. In app.php do the same thing. But since performance is totally rad to have in the prod environment, add an include_once below that for /../var/bootstrap.php.cache:

47 lines | web/app.php

```
... lines 1 - 11
$loader = require __DIR__.'/../app/autoload.php';
include_once __DIR__.'/../var/bootstrap.php.cache';
... lines 14 - 47
```

Let's give that a try in the browser.

And it *still* works. I can't seem to break our app.

Moving Tests

We're now down to the last change: it involves the tests directory. Wait, you don't write tests? You are a brave, brave developer.

For the rest of you law-abiding citizens. the tests *used* to live inside of the bundles themselves. These have now been moved down into a new root directory called, well, tests! Move the AppBundle/Tests directory into tests. Once it's there, rename it to AppBundle so that you have a nice tests/AppBundle.

The idea is to have paths like tests/bundleName/individualClasses. There's no technical reason for this change, it's just a new standard.

With everything moved around, you'll need to update the namespace of each class to be Tests/AppBundle:

41 lines | tests/AppBundle/Controller/BlogControllerTest.php

```
... lines 1 - 11
namespace Tests\AppBundle\Controller;
... lines 13 - 41
```

The reason for this is autoloading: you know, the old "your directory structure must match your namespace" thing. You can sometimes get away with messing this up with tests... but not always.

This is sweet... except that Composer doesn't know to look in the tests/ directory. Open composer.json. Below autoload, create a new section called autoload-dev. Below it add a psr-4 key for { "Tests\\": } pointing at the tests/ directory:

66 lines | composer.json

```
... lines 1 - 9
    "autoload-dev": {
        "psr-4": { "Tests\\": "tests/" }
    },
... lines 13 - 66
```

We're good!

Moving phpunit.xml.dist

On the topic of tests, the phpunit.xml.dist file *normally* lives in the app/ directory. That's why you run phpunit -c app.

To simplify things, that has been moved to the root of the project:

```
$ mv app/phpunit.xml.dist .
```

After moving it, we need to update a few things. Update bootstrap to app/autoload.php. In the testsuites section, this is how phpunit know *where* tests live. This can now simply be changed to tests... which is pretty cool:

41 lines | phpunit.xml.dist

```
... lines 1 - 3
<phpunit
... lines 5 - 13
    bootstrap                = "app/autoload.php" >

    <testsuites>
        <testsuite name="Project Test Suite">
            <directory>tests</directory>
        </testsuite>
    </testsuites>
... lines 21 - 39
</phpunit>
```

Finally, uncomment out the php block and tell Symfony where your app directory is by changing the KERNEL_DIR value to app/:

41 lines | phpunit.xml.dist

```
... lines 1 - 3
<phpunit
... lines 5 - 21
    <php>
        <server name="KERNEL_DIR" value="app/" />
    </php>
... lines 25 - 39
</phpunit>
```

This is for functional tests: when Symfony tries to boot your kernel, it needs to know... where you kernel lives!

Get back to the terminal. But don't run phpunit -c app, that's *old* news. Just run:

```
$ phpunit
```

And hey, our tests are even passing!

Changes to .gitignore

After making all of these changes, git status looks a little crazy:

```
$ git status
```

Open up .gitignore. Ok, this guy is totally out of date now. Let's help him out.

The var directory now completely holds things that you do *not* need to commit to your repository. So, instead of ignoring individual things, ignore the entire /var/ directory. We *do* want to keep the .gitkeep file in cache and logs, but change each to start with /var/. The app/config/parameters.yml file doesn't change and now we need to ignore any phpunit.xml file if it exists. We don't need to ignore /bin/ any longer and I'll remove composer.phar from the list since that's usually installed globally. Keep vendor and web/bundles at the bottom:

8 lines | .gitignore

```
/var/
!var/cache/.gitkeep
!var/logs/.gitkeep
/app/config/parameters.yml
/phpunit.xml
/vendor/
/web/bundles/
```

Ah that looks much better.

Run git status again in the terminal:

```
$ git status
```

SO much better.

Tiny Configuration Changes

Make one last little change in config.yml. Under the framework key, add assets: ~:

104 lines | app/config/config.yml

```
... lines 1 - 20
framework:
... lines 22 - 47
    assets: ~
... lines 49 - 104
```

This guarantees that you can still use the asset() function in twig. We don't need this in 2.8 but in 3.0 you have to turn that on explicitly. This is actually pretty cool: instead of the framework bundle turning on a lot of features automatically, you get to opt into the features you want available.

Okay that is it! Enjoy your brand new fancy directory structure. I really like it because the var directory is stuff I don't need to touch, the app directory is just configuration and the tests/ are all together for the holidays.

Chapter 3: Fix Deprecation Warnings from Bundles

Once you're on 2.8, there's a new game: find and fix deprecation notices. But there's a catch! You won't hit *all* of your code paths at runtime. Some code paths are only executed when the cache is being built. To hit those, start by clearing the cache:

```
$ rm -rf var/cache/*
```

Refresh the page and click into the deprecated calls on the Web Debug Toolbar.

Deprecations come from Outside Bundles

Ok, here's the *really* tricky thing about deprecation notices: many of them are *not your fault*. Yep, a lot of notices come from all of those lazy third-party bundles you're using in your project. So to start, let's find out what outside bundles are causing problems... before we worry about fixing our stuff.

The first deprecation - about not quoting @ symbols in YAML is *our* fault.

> Not quoting a scalar starting with '@' is deprecated since Symfony 2.8

But it's not easy to see: you need to study the stack trace. This ultimately starts with AppKernel::registerContainerConfiguration() where *our* configuration files are loaded.

The second is complaining about bad configuration in security.yml: that's *also* our fault, and we'll fix it in a minute.

But look at the third warning:

> The Symfony\Component\DependencyInjection\Reference::isStrict method is deprecated since version 2.8

Look closely at the LoggerChannelPass: that's coming from MonologBundle. That's the first outside bundle that needs to be updated.

Below that, the knp_pagination Twig extension problem is obviously coming from KnpPaginatorBundle.

But before you upgrade those, go back and refresh again. This time the page pulls from the cache, and that can sometimes cause different deprecation notices to show up. A-ha!

> The class "Symfony\Bundle\AsseticBundle\Config\AsseticResource" is performing resource checking through ResourceInterface::isFresh(), which is deprecated since 2.8

This deprecation warning comes from AsseticBundle: that's our *third* troublesome library.

Updating Bundles with Deprecations

Update these by running composer update followed by their names. Copy them from composer.json:

```
$ composer update knplabs/knp-paginator-bundle symfony/assetic-bundle symfony/monolog-bundle --with-dependencies
```

Notice I'm *not* tweaking their version constraints in composer.json. Maybe I *will* need to do this, but I'll take the lazy route first and hope that upgrading these to the newest version allowed by their existing constraints will be enough.

Let's see what happens!

Cool! This downloaded some patch version updates, which *may* have solved our problem... or maybe it didn't. Clear the cache and go refresh:

```
$ rm -rf var/cache/*
```

Click into the deprecations. Now the notices are down to 5 and they're coming from *our* code. Problem solved! But wait, refresh again. Huh, now there are 32 notices: the ones from AsseticBundle are back! The new version of AsseticBundle did *not* fix that problem.

The Bundle isn't Symfony 3 Ready!?

Google for AsseticBundle and go to its Github Page. The first thing to look for is a release that has Symfony 3 support. Huh, there *is* one that claims support: version 2.7.1. And that *is* the version of the library we just downloaded. Usually, this means you're fine... but *clearly* it's not fine: we're still getting deprecated notices! What's going on AsseticBundle!

In fact, at the time of recording, this bundle claims Symfony 3 support, but it doesn't *quite* have it: there's one pull request that still needs to be merged. By the time you're watching this, it'll hopefully be merged and you'll happily get the 2.7.2 version.

But I'm glad this happened: it uncovers the most difficult part of upgrading to Symfony 3. If you use a lot of outside bundles, they might not all be ready immediately when Symfony 3 is released. In fact, some might not be updated a month later. Your job is to check the repository, see if there is a Symfony 3-compatible release, and open up a friendly issue if there isn't.

For right now, until this is merged, there's nothing we can do. So let's ignore these deprecations and fix everything else.

Chapter 4: Deprecation Fixing Tools

Tip

The video for this tutorial will be ready **very** soon!

Ideally, your outside bundles are no longer triggering deprecated warnings. Now it's time to update *our* code for Symfony 3. This means finding deprecation warnings, fixing them everywhere you can think of, and, well, repeating! It's pretty simple, but sometimes the true source of a deprecation can be tricky to find.

Symfony Upgrade Fixer

Fortunately, there are 2 tools to help us. Google for Symfony Upgrade Fixer. This is a sweet tool made by my good friend Saša. It tries to find deprecated code and fix it *for* you. How nice!? It only supports a few things, but it aims to fix the most *annoying* ones, like form changes.

Copy the wget line and paste it into your command line. Or just use that URL to download the file manually. Copy the second permissions line to make that file executable. Cool! Now we have a symfony-upgrade-fixer utility! Run it!

```
$ symfony-upgrade-fixer
```

And then actually *use* it with fix and then . to fix this directory:

```
$ symfony-upgrade-fixer fix .
```

Behind the scenes, that's snooping through your project and writing new code for you. Ding! It fixed two form classes:

80 lines | src/AppBundle/Form/PostType.php

```
... lines 1 - 25
class PostType extends AbstractType
{
... lines 28 - 30
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
... lines 33 - 42
        $builder
... lines 44 - 47
            ->add('summary', TextareaType::class, array('label' => 'label.summary'))
            ->add('content', TextareaType::class, array(
... lines 50 - 51
            ))
            ->add('authorEmail', EmailType::class, array('label' => 'label.author_email'))
... lines 54 - 55
            ))
        ;
    }
... lines 59 - 78
}
```

75 lines | src/AppBundle/Form/Type/DateTimePickerType.php

```
... lines 1 - 27
class DateTimePickerType extends AbstractType
{
... lines 30 - 61
    public function getParent()
    {
        return DateTimeType::class;
    }
... lines 66 - 73
}
```

By using git diff, I can see that it updated to the new TextareaType::class format. If you have a lot of forms, this is *awesome*.

Symfony Deprecation Detector

This tool caught a *few* things, but there's a lot more. So... let's try another tool! Google for Symfony Deprecation Detector. This tool doesn't fix your project, but it detects more things than the first one.

Copy the clone command and run it in the terminal:

```
$ cd any/path
$ git clone git@github.com:sensiolabs-de/deprecation-detector.git
```

Move into the directory and install the composer libraries:

```
$ cd deprecation-detector
$ composer install
```

This will give us a new binary that we can use on our project. Switch back into the directory of *our* project, then run the bin/deprecation-detector command from that project:

```
$ /path/to/deprecation-detector/bin/deprecation-detector
```

Boom! This correctly sees that 3 form types still have a getName() method, which they should *not* have anymore. Go into AppBundle/Form, open up each form type, and remove the getName() method from the bottom.

Perfect! Try the command again:

```
$ /path/to/deprecation-detector/bin/deprecation-detector
```

No violations! I *wish* this meant that you had no more deprecated features, but it's an imperfect tool. We need to hunt for the last few deprecations.

Chapter 5: Fix My Deprecations

The last step is to find and eliminate the final deprecation warnings. Since the changes to the form system are some of the biggest, navigate to a page with a form. Woh, error!

> The type name specified for the service "app.form.type.datetimepicker" does not match the actual name.

Because the way forms are named has changed, I know this has *something* to do with that. The stack trace points us to BlogController line 149:

220 lines | src/AppBundle/Controller/Admin/BlogController.php

```
... lines 1 - 70
71      public function newAction(Request $request)
72      {
... lines 73 - 76
77          $form = $this->createForm(new PostType(), $post)
... lines 78 - 109
110     }
... lines 111 - 220
```

And that line points us to PostType. A-ha! The publishedAt field type is still app_datetimepicker:

80 lines | src/AppBundle/Form/PostType.php

```
... lines 1 - 25
26  class PostType extends AbstractType
27  {
... lines 28 - 30
31      public function buildForm(FormBuilderInterface $builder, array $options)
32      {
... lines 33 - 42
43          $builder
... lines 44 - 53
54              ->add('publishedAt', 'app_datetimepicker', array(
55                  'label' => 'label.published_at',
56              ))
57          ;
58      }
... lines 59 - 78
79  }
```

The upgrade fixer did *not* fix this type name, because it's custom. Change this to DateTimePickerType::class. Then go back, hit option+enter, and click "Import class" to add the use statement:

71 lines | src/AppBundle/Form/PostType.php

```
... lines 1 - 31
32      public function buildForm(FormBuilderInterface $builder, array $options)
33      {
... lines 34 - 54
55              ->add('publishedAt', DateTimePickerType::class, array(
... line 56
57              ))
58          ;
59      }
... lines 60 - 71
```

While we're here, BlogController has a problem too. Instead of saying new PostType(), you need to say PostType::class:

220 lines | src/AppBundle/Controller/Admin/BlogController.php

```
... lines 1 - 220
```

Refresh! It works! Open the deprecation notices. There are still 3: one for the problematic AsseticBundle, one for unquoted @ symbols and another because the global _self variable in Twig is deprecated. The _self usage comes from the Symfony Demo itself: it's used in the "Show Code" functionality. Since that's specific to the Symfony Demo, I'll ignore that one.

Finding Unquoted @'s

So let's keep going. Open up app/config/services.yml. Starting in 2.8, you need to surround any string that starts with @ with quotes. Yea, it turns out that @ is a special character, so it was *always* illegal to have unquoted @ symbols, but the YAML parser was nice and worked anyways. Well, now the YAML parser is super strict, so surround @markdown with single quotes. Do the same around @router:

51 lines | app/config/services.yml

```
1   services:
... lines 2 - 11
12      app.twig.app_extension:
... lines 13 - 14
15          arguments: ['@markdown', %app_locales%]
... lines 16 - 24
25      app.redirect_to_preferred_locale_listener:
... line 26
27          arguments: ['@router', %app_locales%, %locale%]
... lines 28 - 51
```

And while you're here, remove the alias from the form.type tag: that's not needed in 3.0:

51 lines | app/config/services.yml

```
1   services:
... lines 2 - 30
31      app.form.type.datetimepicker:
32          class: AppBundle\Form\Type\DateTimePickerType
33          tags:
34              - { name: form.type }
... lines 35 - 51
```

Let's see what that did. Go back and clear the cache:

```
$ rm -rf var/cache/*
```

Refresh! Now we're down to *4* deprecations, and two are *still* coming from unquoted YAML files. Hmm. Look at the stack trace. It's not easy to figure out *where* this problem is coming from. But look: CodeExplorerExtension::load(): that class lives in *our* code: inside the CodeExplorerBundle. Open that class:

31 lines | src/CodeExplorerBundle/DependencyInjection/CodeExplorerExtension.php

```
... lines 1 - 21
class CodeExplorerExtension extends Extension
{
    public function load(array $configs, ContainerBuilder $container)
    {
        $loader = new YamlFileLoader($container, new FileLocator(__DIR__.'/../Resources/config'));

        $loader->load('services.yml');
    }
}
```

Inside, we load a services.yml file. Ah, I bet there's an @ symbol inside! And there it is: wrap that string in single quotes:

14 lines | src/CodeExplorerBundle/Resources/config/services.yml

```
services:
... lines 2 - 8
    code_explorer.controller_listener:
        class:     CodeExplorerBundle\EventListener\ControllerListener
        arguments: ['@code_explorer.twig.source_code_extension']
... lines 12 - 14
```

Let's do it again: clear the cache and refresh:

```
$ rm -rf var/cache/*
```

But there's still *one* unwrapped @ symbol. Look *really* closely. This time, you can see that the trace starts with a Router::getRouteCollection() followed by a YamlFileLoader. This makes me think that this is coming from a routing.yml file. Open the main one. Of course! Put quotes around the @AppBundle string:

28 lines | app/config/routing.yml

```
... lines 1 - 6
app:
    resource: '@AppBundle/Controller/'
... lines 9 - 28
```

Configuration Tweaks

Before you refresh, fix one more: the csrf notice. In this case, a config key was renamed from csrf_provider to csrf_token_generator in security.yml. A lot of changes are like this: simple renames:

46 lines | app/config/security.yml

```
security:
... lines 2 - 14
    firewalls:
        secured_area:
... lines 17 - 25
            form_login:
... lines 27 - 33
                csrf_token_generator: security.csrf.token_manager
... lines 35 - 46
```

Clear the cache and refresh:

```
$ rm -rf var/cache/*
```

And boom! The only deprecation left *is* coming from our code, but I'm going to ignore it because fixing it isn't very interesting.

Refresh one more time to simulate using the cached files. We can still see the *other* problem we have because AsseticBundle isn't yet compatible with 3.0.

In real life, this *will* block us from continuing to Symfony 3.0. Outside bundles will be the biggest issue with upgrading. But don't worry! Until then, you have all the features you need on 2.8. Be patient.

In your project, continue going through other sections of your site, looking for - and fixing - more deprecated code. And of course, run your tests: if you have the phpunit bridge installed, you'll get a print-out of deprecated calls hit from your tests.

Once the deprecations are gone: you're ready for 3.0.

Chapter 6: Upgrade Outdated Libraries!

If you did you homework, then by now, you've gone through your entire site and updated your code to remove all of the deprecation warnings.

Upgrading Offending Libraries/Bundles!

Except for the ones that you *can't* fix, because they're not coming from your code. It's pretty likely that you have a group of external libraries and bundles that are *also* causing deprecation warnings.

In our app, we have this exact problem, with AsseticBundle. So before we upgrade to Symfony 3, we need to upgrade to a new version of that library that does *not* use old code.

No problem! Google for AsseticBundle. I'll click into the link on Packagist. I'm looking here to find the latest released version of Packagist, which right now, is 2.8.0. If we update to that, those deprecation warnings will *hopefully* go away.

Open up composer.json:

```
66 lines | composer.json
1   {
    ... lines 2 - 12
13      "require": {
    ... lines 14 - 27
28          "symfony/assetic-bundle"        : "~2.6",
    ... lines 29 - 32
33      },
    ... lines 34 - 64
65  }
```

Our version is ~2.6, but my PhpStorm Composer plugin tells me that we actually have 2.7.1. Let's require 2.8.0 by changing this to ^2.8:

```
66 lines | composer.json
1   {
    ... lines 2 - 12
13      "require": {
    ... lines 14 - 27
28          "symfony/assetic-bundle"        : "~2.8",
    ... lines 29 - 32
33      },
    ... lines 34 - 64
65  }
```

The ^ symbol is the new hipster way to describe your version constraints - it basically allows 2.8 and higher, but not some future 3.0 version of AsseticBundle.

Next, copy the library name - symfony/assetic-bundle. Since we *only* want to upgrade this library right now, find your terminal and run:

```
$ composer update symfony/assetic-bundle
```

What Version to Choose?

In general, if a library is giving you deprecation warnings like this one, you *do* need to upgrade it, but you may *not* need to upgrade it to the absolute latest version. If you want to be a bit more conservative, you can dig into the release notes of a library to find the *first* version number that is Symfony 3 compatible. It's more work, but might mean less QA time than

upgrading a library many versions all at once.

Hello AsseticBundle 2.8.0! Oh, and see those messages from Composer? That's another usage of deprecations in action: some of our code is using some deprecated functionality in Composer itself. Those will probably go away when we upgrade Symfony. But, either way, it's not important right now.

Let's check in on our deprecations: refresh the page. Yes! 33 deprecations down to *one*. And don't worry about that last deprecation - it's really specific to this project - so I'm not going to take time to fix it.

Ok, we've now upgraded all outside libraries that were not Symfony 3 compatible and our deprecation warnings are gone. We are ready.

Chapter 7: Upgrading to Symfony 3!

We are now finally ready to upgrade our project to Symfony 3. How do we do that? It's a simple, 2 step process.

First, in composer.json change the symfony/symfony key, to 3.0.*:

```
66 lines | composer.json
{
... lines 2 - 12
    "require": {
... lines 14 - 30
        "symfony/symfony"                    : "3.0.*",
... line 32
    },
... lines 34 - 64
}
```

Or, if you're upgrading to any even new version, use that - like 3.1.*.

Second, run composer update. This time, run it with *no* arguments:

```
$ composer update
```

We technically *only* need to upgrade the symfony/symfony package, but since *so* many libraries depend on this, it's pretty tough to *only* update Symfony. If you're worried about too much stuff upgrading, make sure you version constraints in composer.json are really *tight*.

But as we wait for this: I have a surprise! This will almost definitely not work.

Upgrading *other* Libraries

And just as I say that, *huge* error! Oh boy, these are a pain to read at first. But if you follow down, eventually you'll find the problem. Ah, there it is: the installation for sensio/distribution-bundle requires symfony/process at version 2.2. In normal English, Composer is saying:

> Yo! Your version of sensio/distribution-bundle in composer.json needs symfony/process 2.2. You probably need to *upgrade* the distribution bundle to a new version that works with symfony/process version 3.

This means we need to update the version for sensio/distribution-bundle in composer.json to something higher. In these cases, I like to open Packagist and search for symfony/framework-standard-edition. This is the project you get when you first download Symfony. We can cheat by looking at *its* composer.json versions.

Let's see what it looks like at the latest 3.0 version - 3.0.8. Ok - this project requires sensio/distribution-bundle at version ^5.0. Open our composer.json and change it to match: ^5.0:

```
66 lines | composer.json
{
... lines 2 - 12
    "require": {
... lines 14 - 25
        "sensio/distribution-bundle"         : "^5.0",
... lines 27 - 32
    },
... lines 34 - 64
}
```

That's it! Run Composer update again:

```
$ composer update
```

We *may* get another error about *another* library, but this is the process: run $ composer update, find the problematic package, update its version and repeat.

Yep: another problem. This time it's from sensio/generator-bundle. Go back to Packagist: the version in the Standard Edition is ^3.0. Update our composer.json:

66 lines | composer.json

```
{
... lines 2 - 33
    "require-dev": {
        "sensio/generator-bundle": "^3.0"
    },
... lines 37 - 64
}
```

And run Composer again:

```
$ composer update
```

If you get an error from a package that's *not* part of the Standard Edition, then you'll need to go look up that library in Packagist or on GitHub to find a version that's 3.0-compatible. That's exactly what we did with AsseticBundle.

Et voila! We just downloaded Symfony 3.0.8.

Well, let's see if it works! Refresh! Yes! Everything looks great: a major framework upgrade *without* breaking your code! Give yourself a congrats. And then, don't forget to run your tests and QA your site to make sure we didn't missing anything.

All right, guys, if you have any questions, let me know.

Seeya next time!