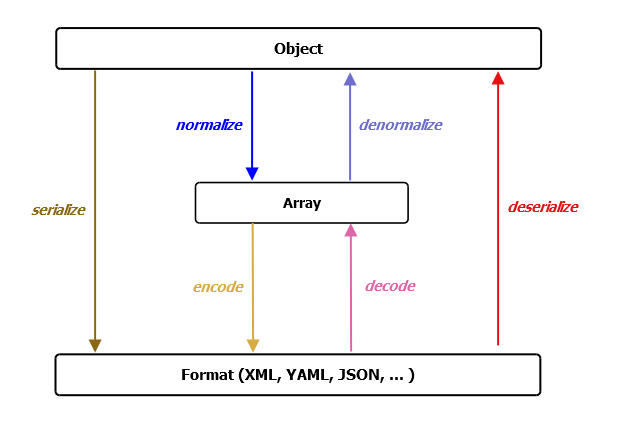
Symfony components in Drupal 8.x

The following are the Symfony components that are going to power the Drupal 8 core to a major extent:

* [**HttpKernel**](https://github.com/symfony/HttpKernel) and **[HttpFoundation](https://github.com/symfony/HttpFoundation)** – These are responsible for streamlining the process of converting a Request into a Response with the help of EventDispatcher. Drupal 8 being moved to Symfony was driven by *Web Services and Content Core Initiative (WSCCI)* in a motive to transform Drupal from a *first-class CMS* to a *first-class* *REST server* with first-class CMS running on top of it. This initiative is intended to allow Drupal to use web services to offer its content with reduced complexity; considering this as a long term vision, Drupal will be more flexible, robust and easily maintainable CMS.
* [**EventDispatcher**](https://github.com/symfony/EventDispatcher) – Implements the [Mediator pattern](http://en.wikipedia.org/wiki/Mediator_pattern) (that uses encapsulation) in a simple and effective manner especially where code inheritance doesn’t work out, making the application highly extensible. It is more effective in situations where you tend to maintain and/or refactor a program consisting of a huge number of classes due to the fact that it makes the communication amongst the classes quite simple and easy.
* [**ClassLoader**](https://github.com/symfony/ClassLoader) – Provides tools that autoload classes and caches their location. PHP uses the[autoloading mechanism](http://php.net/manual/en/language.oop5.autoload.php) to delegate the loading of a file that defines the class in situations where you reference a class that has not been required or included yet. Symfony comes with autoloaders likePSR-0 Class Loader and MapClassLoader. Implementing the ClassLoader component will make Drupal module developers carefree especially when it comes to implementing module\_load\_includeand other dependencies. Moreover, it allows developers easy calling of classes during run-time.
* [**YAML**](https://github.com/symfony/Yaml) – It parses YAML strings and converts them to PHP arrays and vice versa. This format has been especially designed to hold configuration related information, while being as expressive as XML files and as readable as INI files. It serves as an integral component of [Drupal’s CMI](https://groups.drupal.org/build-systems-change-management/cmi) (Configuration Management Initiative) that allows our modules to initially define their default configuration settings and later allows the site builder to override the same as-and-when instructed to. This concept of Drupal 8’s CMI which is powered by YAML is a replacement for [Features contributed Drupal module](https://drupal.org/project/features)which proves to be a robust concept as far as migrating and deploying across environments is concerned.
* [**Routing**](https://github.com/symfony/Routing) – Allows us to load all routes, and dumps a URL matcher or generator specific to these routes. This also means that it maps an HTTP request to a set of configuration variables. As far as Drupal 8 and above versions are concerned, we define our module’s routes in a YAML configuration file, each of them set to trigger a specific action that has been defined in our module’s classes.
* [**DependencyInjection**](https://github.com/symfony/DependencyInjection) – Primarily used to standardize and centralize the way objects are constructed in our application. Symfony’s DependencyInjection component has been incorporated in Drupal 8 in an intention to write code in such a way that classes can be reused and unit-tested wherever applicable and desired.
* [**Twig**](http://twig.sensiolabs.org/) – Drupal 8 has adopted the [Twig](https://drupal.org/node/1499460) template engine. This is of interest to the themers who will probably never think of any other option again for as long as they’re working on Drupal themes. Twig was developed by *Fabien Potencier*, who also developed the [Symfony project](https://drupal.org/project/symfony), and was fine tuned for integration into Drupal 8.
* [**Process**](https://github.com/symfony/Process) – Helps execute commands in sub-processes using the command-line interface. Drupal 8 will use this to handle all activities that are command-line in nature.
* [**Serializer**](https://github.com/symfony/Serializer) – It is used to transform objects into a specific format (eg. XML, YAML, JSON, etc.) and vice versa. To understand it better, let us look at the following schema that a *Serializer* component follows:

Additionally, we can use it to accomplish a number of jobs, ranging from configuration to node and entity creation that should be delivered by a REST endpoint.

* [**Validator**](https://github.com/symfony/Validator) – Helps Drupal validate values. For example: validating form submission, validating entities within Drupal, etc. To accomplish its job, it uses *Doctrine Annotations* (discussed in *Out-of-the-box third party components* section).
* [**Translation**](https://github.com/symfony/Translation) – Provides a standard set of tools to load translation files, generate translated strings as output, and use the generated outcome.

Out-of-the-box third party components

We went through Symfony components that have been included with Drupal 8. In this section, we’ll find out about the 3rd party PHP components that have been implemented in Drupal 8 and are not related to Symfony. They are:

* [**Assetic**](https://github.com/kriswallsmith/assetic) – An *Asset Management framework*. Assets in Drupal consist of CSS files, JavaScript files, images, other media files, metadata, etc. These assets contain filterable content that can be loaded and dumped. In order to manipulate assets, you need to apply appropriate filters. To understand the filters, in what situations each can be used, how-to use the filters, build, dump, and much more, we recommend you read the [Assetic Github page](https://github.com/kriswallsmith/assetic).
* [**Composer**](http://www.sitepoint.com/php-dependency-management-with-composer/) – A tool specifically designed and developed to *manage dependency* in PHP allowing us to declare the dependent libraries our project needs and install them for us. Though it deals in packages and libraries, it is not a package manager. Composer is a boon to Drupal 8 due the following:
  + Handles situations very efficiently wherein your project is dependent upon a number of libraries.
  + Tackles situations where there is a nested dependency concept amongst the libraries. For instance; your project is dependent upon a library (say, *lib1*) and in tern *lib1* is dependent upon some other library (say, *lib2*).
  + It is Composer’s responsibility to choose which version of the package or library needs to be installed unless explicitly told which versions to target.
* [**Doctrine**](http://www.doctrine-project.org/) – A bundle of tools for use with database records. Drupal 8 uses a part of it called*Annotations* that exposes additional metadata to Drupal. Efforts have been made to present such tricky functionality in a quite easy and handy way, especially when defining custom entity types. If you wish to explore further on how Doctrine is supposed to be used with Drupal 8, we recommend you visit the [documentation page](https://drupal.org/node/1817778).
* [**EasyRDF**](http://www.easyrdf.org/) – A PHP library that has been used with Drupal 8 so as to provide ease in consuming and producing RDF. With the help of EasyRDF, Drupal 8 adds capability to produce metadata in the markup in an easy and convenient way. You may also visit [EasyRDF official site](http://www.easyrdf.org/) to learn more about it.
* [**Guzzle**](http://www.sitepoint.com/guzzle-php-http-client/) – It is a PHP HTTP client and framework that helps Drupal 8 make web requests using REST based web service calls. This makes Drupal 8 web portals more efficient in terms of handling different sorts of web services. To explore further on Guzzle, we recommend [reading our previous article on it](http://www.sitepoint.com/guzzle-php-http-client/).
* [**PHPUnit**](http://www.sitepoint.com/getting-started-with-phpunit/) – Used in Drupal 8 to standardize unit testing throughout the core as well as help developers write code that is of utmost quality. PHPUnit ensures that any code written in Drupal 8 and in any custom module incorporated in it matches industry standards and works as expected throughout its life-cycle. You may keep an eye on the current status of PHPUnit’s incorporation into Drupal 8 by going[here](https://drupal.org/node/1901670).
* [**PSR-3 Logging**](http://www.sitepoint.com/logging-with-psr-3-to-improve-reusability/) – A common logging system that is shared by an entire PHP application. Drupal 7 and older versions use watchdog() for this purpose. Switching Drupal logging from watchdog() to a PSR-3 logging framework has made Drupal 8 more robust and scalable in terms of common logging. You may track the current status of this component at [Switch watchdog to a PSR-3 logging framework](https://drupal.org/node/1289536).

Conclusion

The introduction of a few Symfony components has simplified working with Drupal 8. Additionally, including a few of the non-Symfony components has added value to Drupal 8 in terms of its quality, scalability, and performance, making it even more robust and maintainable.