UmlWriter Book

Laurent Laville

UmlWriter Book

Laurent Laville

Table of Contents

This complete guide documents UmlWriter 1.1.0, published on 2015-12-09.

Part I. Getting Started

Chapter 1. Download

We distribute a PHP Archive [http://www.php.net/phar] (PHAR) that contains all required dependencies of UmlWriter bundled in a single file.

Download the latest version [http://bartlett.laurent-laville.org/get/umlwriter-1.1.0.phar]

Make it executable, and put it into your `$PATH`.

```
$ chmod +x umlwriter-1.1.0.phar
$ mv umlwriter-1.1.0.phar /usr/local/bin/umlwriter

$ umlwriter --version
```

You can also immediately use the PHAR after you have downloaded it.

```
$ wget http://bartlett.laurent-laville.org/get/umlwriter-1.1.0.phar
$ php umlwriter-1.1.0.phar --version
```

With both methods then you have this output :

```
umlWriter version 1.1.0
```

Chapter 2. Generating a Diagram

UML diagrams produced, may be either :

- restricted to a simple class and its direct dependencies.
- restricted to a single namespace (with all its objects).
- non-restricted (all namespaces and their objects).

The following diagram processors (output formats) are available:

- Graphviz syntax with HTML-like labels
- PlantUML syntax with default fonts and colors

Detailed descriptions and examples of the diagramming languages are available on the Graphviz [http://graphviz.org/] and PlantUML [http://plantuml.sourceforge.net/] websites.

The following reverse-engines are supported:

- Bartlett\Reflect [https://github.com/llaville/php-reflect] (default in CLI mode)
- Andrewsville\TokenReflection [https://github.com/Andrewsville/PHP-Token-Reflection]

Chapter 3. Generating a diagram statements from a Terminal

- In graphviz format with `bartlett/php-reflect` reverse-engine

  ```
  $ umlwriter diagram:render --reflector=reflect --processor=graphviz /path/to/data_sourc
  ```

- In graphviz format with `andrewsville/php-token-reflection` reverse-engine

  ```
  $ umlwriter diagram:render --reflector=tokenreflection --processor=graphviz /path/to/da
  ```

- In plantuml format with default reverse-engine (`bartlett/php-reflect`)

  ```
  $ umlwriter diagram:render --processor=plantuml /path/to/data_source
  ```

Chapter 4. Generating a diagram image

You must have installed correctly graphviz or plantuml before to continue.

- With Graphviz and `png` output format

  ```
  $ dot -Tpng -O /path/to/gv_file
  ```

  Where `/path/to/gv_file` is a file contening Graphviz syntax statements produced by the umlwriter command.

- With PlantUML and `png` output format

  ```
  $ java -jar plantuml.jar -Tpng /path/to/puml_file -o /path/to/ouput/dir
  ```

  Where `/path/to/puml_file` is a file contening PlantUML syntax statements produced by the umlwriter command.

  And `/path/to/output/dir` is the directory where the image will be generated.

Chapter 5. Summary

Let’s review what we’ve done :

- downloaded the latest stable PHAR version.
- created project using Composer.
- built your first graphviz and plantuml UML diagram (and `png` image) from any data source.

Chapter 6. Next

Choose your way depending of your skill level.

Read more

- Want to learn more on CLI tool, have a look on Part II, “User Guide”
- Want to learn more about the Graphviz or PlantUML processor, have a look on Part III, “Processor Guide”
- Want to learn more about the compatible reverse-engine, have a look on Part IV, “Reflector Guide”

Part II. User Guide

First visit, you are highly recommended to follow chapters in following order.

1. Installing all necessary UmlWriter components. See Chapter 7, *Installation*
2. Running your first parses with the Command-Line interface. See Chapter 8, *The Command-Line*

Chapter 7. Installation

UmlWriter may be installed in several ways, choose your favorite.

7.1. Requirements

Before you install PHP UmlWriter, you will need an operating system with PHP [http://www.php.net] 5.4.0 or later installed,

UmlWriter requires the spl [http://www.php.net/manual/en/book.spl.php] extensions. This extension is usually compiled and enabled by default.

7.2. Composer

Put a file named composer.json at the root of your project, with the content below:

```
{
    "require": {
        "bartlett/umlwriter": "1.1.0"
    }
}
```

And ask Composer [http://getcomposer.org/] to install the dependencies:

```
$ php composer.phar install
```

Or just invoke Composer to install the latest version:

```
$ php composer.phar require bartlett/umlwriter
```

You can also use Composer to create a new project from an existing UmlWriter package. This is the equivalent of doing a git clone checkout followed by a composer install of the vendors.

```
$ php composer.phar create-project bartlett/umlwriter /path/to/install 1.1.0
```

Where `/path/to/install` is your install directory.

7.3. PHAR

The recommended way for newbies, or just to have a look on features of this library, is to download a PHP Archive that contain all required dependencies of PHP UmlWriter bundled in a single file.

```
$ wget http://bartlett.laurent-laville.org/get/umlwriter-1.1.0.phar
$ chmod +x umlwriter-1.1.0.phar
$ mv umlwriter-1.1.0.phar /usr/local/bin/umlwriter
$ umlwriter
```

You can also immediately use the PHAR after you have downloaded it.

```
$ wget http://bartlett.laurent-laville.org/get/umlwriter-1.1.0.phar
$ php umlwriter-1.1.0.phar
```

Chapter 8. The Command-Line

The command-line interface is the easiest way to try and learn the basic UmlWriter features.

For all users.

8.1. Command-Line Options

When you invoke the `umlwriter` command, you should obtain the following commands and options :

```
                  ___        __    _ _
  _   _ _ __ ___ | \ \      / / __(_) |_ ___ _ __
 | | | | '_ ` _ \| |\ \ /\ / / '__| | __/ _ \ '__|
 | |_| | | | | | | | \ V  V /| |  | | ||  __/ |
  \__,_|_| |_| |_|_|  \_/\_/ |_|  |_|\__\___|_|

umlWriter version 1.1.0

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display this help message
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi            Force ANSI output
      --no-ansi         Disable ANSI output
  -n, --no-interaction  Do not ask any interactive question
      --manifest            Show which versions of dependencies are bundled.
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for m

Available commands:
  help                      Displays help for a command
  list                      Lists commands
 diagram
  diagram:render            Generate diagram statements of all objects
  diagram:render:class      Generate diagram statements of a single class
  diagram:render:namespace  Generate diagram statements of a single namespace
```

diagram:renderGenerate diagram statements of all objects.

```
$ umlwriter diagram:render --processor=graphviz src/Bartlett/UmlWriter/Proce
```

```
digraph G {
        overlap = false;
        node [fontname="Verdana", fontsize="8", shape="none", margin="0", fi
        edge [fontname="Verdana", fontsize="8"];
        subgraph cluster_0 {
                label="Bartlett\\UmlWriter\\Processor";
                "Bartlett\\UmlWriter\\Processor\\AbstractProcessor" [label=<
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>abstract</i></font> &
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">#  reflector</td></tr>
<tr><td align="left">#  graphId</td></tr>
```

```
<tr><td align="left"># objects</td></tr>
<tr><td align="left"># edges</td></tr>
<tr><td align="left">- spaces</td></tr>
<tr><td align="left">- linebreak</td></tr>
</table></td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">+__construct()</td></tr>
<tr><td align="left">+setGraphId()</td></tr>
<tr><td align="left">+renderClass()</td></tr>
<tr><td align="left">+renderNamespace()</td></tr>
<tr><td align="left">+render()</td></tr>
<tr><td align="left">#renderEdges()</td></tr>
<tr><td align="left">#formatLine()</td></tr>
<tr><td align="left">#writeObjectElement()</td></tr>
<tr><td align="left">#writeObjectInheritance()</td></tr>
<tr><td align="left">#writeObjectInterfaces()</td></tr>
<tr><td align="left">#writeConstantElements()</td></tr>
<tr><td align="left">#writePropertyElements()</td></tr>
<tr><td align="left">#writeMethodElements()</td></tr>
<tr><td align="left">#<i>pushObject</i>()</td></tr>
<tr><td align="left">#<i>pushEdge</i>()</td></tr>
<tr><td align="left">#formatClassStereotype()</td></tr>
<tr><td align="left">-reset()</td></tr>
</table></td></tr>
</table>
>];
                "Bartlett\\UmlWriter\\Processor\\GraphvizProcessor" [label=<
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>class</i></font> &gt;
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">># namespaceSeparator</td></tr>
</table></td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">#renderObjects()</td></tr>
<tr><td align="left">#renderEdges()</td></tr>
<tr><td align="left">#writeGraphHeader()</td></tr>
<tr><td align="left">#writeGraphFooter()</td></tr>
<tr><td align="left">#pushObject()</td></tr>
<tr><td align="left">#pushEdge()</td></tr>
<tr><td align="left">-attributes()</td></tr>
</table></td></tr>
</table>
>];
                "Bartlett\\UmlWriter\\Processor\\ProcessorInterface" [label=
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>interface</i></font>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">+ GLOBAL_NAMESPACE</td></tr>
</table></td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">+renderClass()</td></tr>
<tr><td align="left">+renderNamespace()</td></tr>
<tr><td align="left">+render()</td></tr>
</table></td></tr>
</table>
>];
                "Bartlett\\UmlWriter\\Processor\\PlantUMLProcessor" [label=<
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>class</i></font> &gt;
```

```
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left"># namespaceSeparator</td></tr>
</table></td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">#renderObjects()</td></tr>
<tr><td align="left">#renderEdges()</td></tr>
<tr><td align="left">#writeGraphHeader()</td></tr>
<tr><td align="left">#writeGraphFooter()</td></tr>
<tr><td align="left">#pushObject()</td></tr>
<tr><td align="left">#pushEdge()</td></tr>
</table></td></tr>
</table>
>];
        }
        "Bartlett\\UmlWriter\\Processor\\GraphvizProcessor" -> "Bartlett\\Um
        "Bartlett\\UmlWriter\\Processor\\GraphvizProcessor" -> "Bartlett\\Um
        "Bartlett\\UmlWriter\\Processor\\PlantUMLProcessor" -> "Bartlett\\Um
        "Bartlett\\UmlWriter\\Processor\\PlantUMLProcessor" -> "Bartlett\\Um
}
```

That may render something like

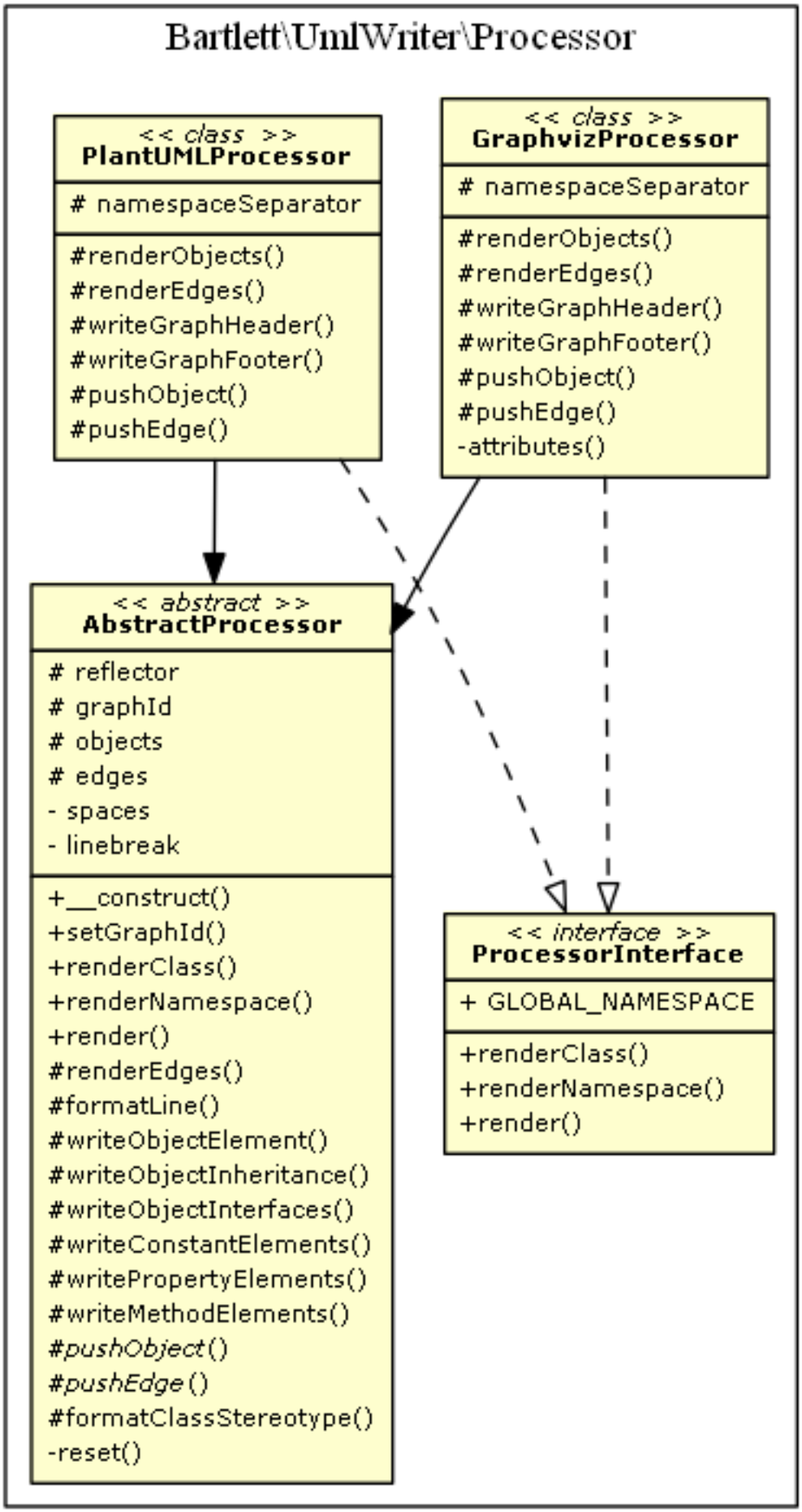

Bartlett\UmlWriter\Processor
<< class >>
PlantUMLProcessor
namespaceSeparator
#renderObjects()
#renderEdges()
#writeGraphHeader()
#writeGraphFooter()
#pushObject()
#pushEdge()
<< class >>
GraphvizProcessor
namespaceSeparator
#renderObjects()
#renderEdges()
#writeGraphHeader()
#writeGraphFooter()
#pushObject()
#pushEdge()
-attributes()
<< abstract >>
AbstractProcessor
reflector
graphId
objects
edges
- spaces
- linebreak
+__construct()
+setGraphId()
+renderClass()
+renderNamespace()
+render()
#renderEdges()
#formatLine()
#writeObjectElement()
#writeObjectInheritance()
#writeObjectInterfaces()
#writeConstantElements()
#writePropertyElements()
#writeMethodElements()
#pushObject()
#pushEdge()
#formatClassStereotype()
-reset()
<< interface >>
ProcessorInterface
+ GLOBAL_NAMESPACE
+renderClass()
+renderNamespace()
+render()

diagram:render:class Generate diagram statements of a single class.

```
$ umlwriter diagram:render:class --processor=graphviz vendor/bartlett/php-re
```

```
digraph G {
        overlap = false;
        node [fontname="Verdana", fontsize="8", shape="none", margin="0", fi
        edge [fontname="Verdana", fontsize="8"];
        subgraph cluster_0 {
                label="Bartlett";
                "Bartlett\\Reflect" [label=<
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>class</i></font> &gt;
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">- analysers</td></tr>
<tr><td align="left">- dataSourceId</td></tr>
</table></td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">+__construct()</td></tr>
<tr><td align="left">+addAnalyser()</td></tr>
<tr><td align="left">+getAnalysers()</td></tr>
<tr><td align="left">+setDataSourceId()</td></tr>
<tr><td align="left">+getDataSourceId()</td></tr>
<tr><td align="left">+parse()</td></tr>
</table></td></tr>
</table>
>];
        }
        subgraph cluster_1 {
                label="Bartlett\\Reflect\\Event";
                "Bartlett\\Reflect\\Event\\AbstractDispatcher" [label=<
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>class</i></font> &gt;
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left"># eventDispatcher</td></tr>
</table></td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">+setEventDispatcher()</td></tr>
<tr><td align="left">+getEventDispatcher()</td></tr>
<tr><td align="left">+dispatch()</td></tr>
<tr><td align="left">+addSubscriber()</td></tr>
</table></td></tr>
</table>
>];
                "Bartlett\\Reflect\\Event\\DispatcherInterface" [label=<
<table border="0" cellborder="1" cellspacing="0">
<tr><td align="center">&lt;&lt; <font color="black"><i>interface</i></font>
<tr><td> </td></tr>
<tr><td><table border="0" cellspacing="0" cellpadding="2">
<tr><td align="left">+setEventDispatcher()</td></tr>
<tr><td align="left">+getEventDispatcher()</td></tr>
<tr><td align="left">+dispatch()</td></tr>
<tr><td align="left">+addSubscriber()</td></tr>
</table></td></tr>
</table>
>];
        }
        "Bartlett\\Reflect\\Event\\AbstractDispatcher" -> "Bartlett\\Reflect
        "Bartlett\\Reflect" -> "Bartlett\\Reflect\\Event\\AbstractDispatcher
```

```
}
```

That may render something like

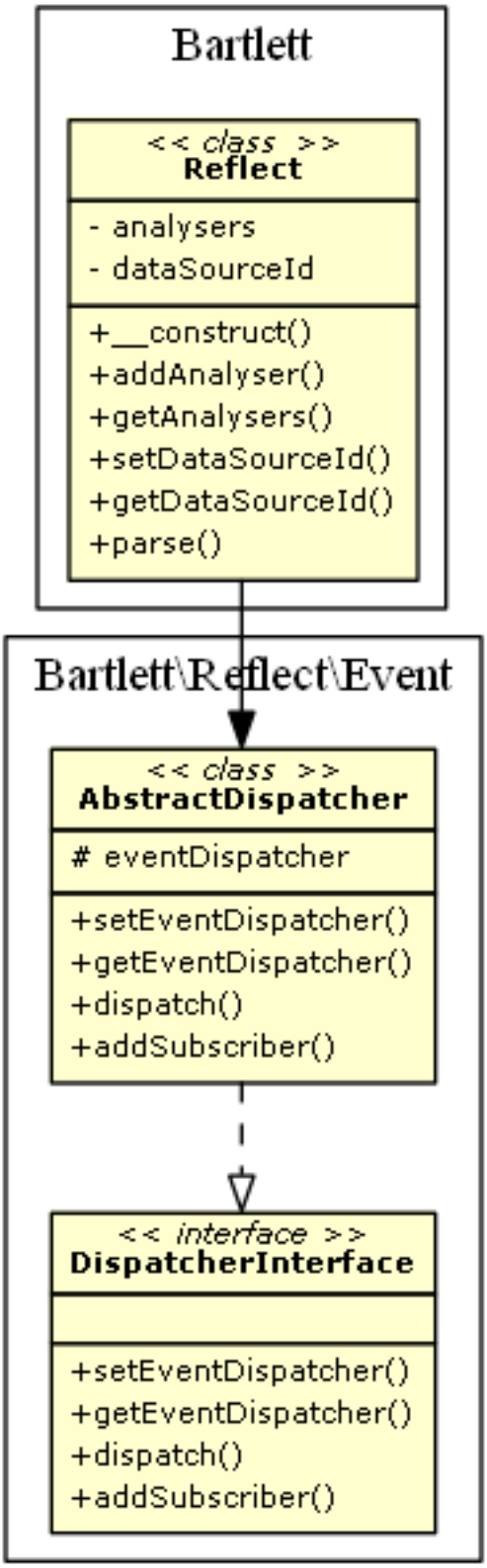
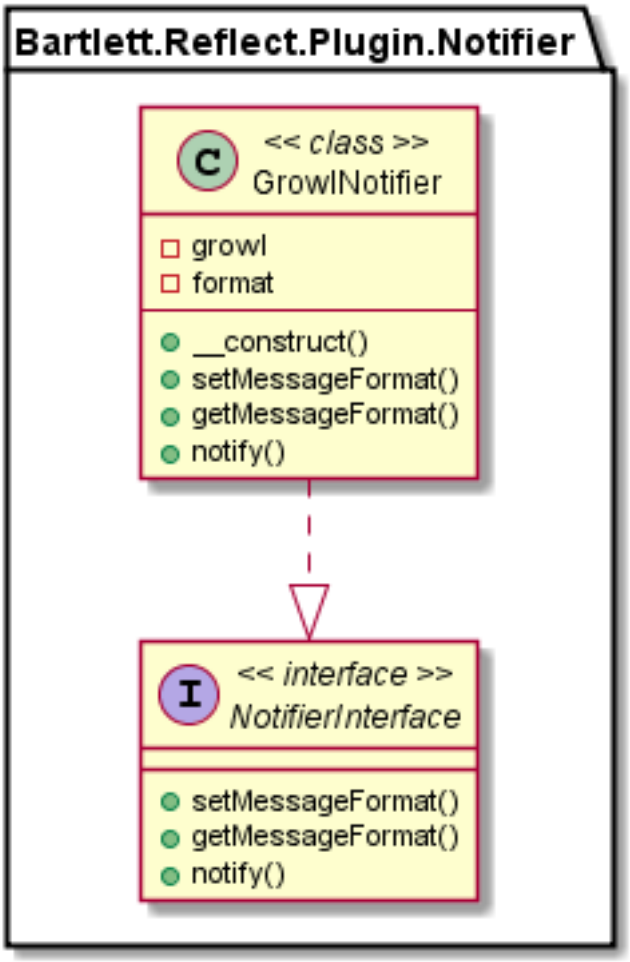


diagram:renderNamespace Generate diagram statements of a single namespace.

```
$ umlwriter diagram:render:namespace --processor=plantuml vendor/bartlett/ph
```

```
@startuml
namespace Bartlett.Reflect.Plugin.Notifier  {
        class GrowlNotifier << class >> {
                -growl
                -format
                --
                +__construct()
                +setMessageFormat()
                +getMessageFormat()
                +notify()
        }
        interface NotifierInterface << interface >> {
                +setMessageFormat()
                +getMessageFormat()
                +notify()
        }
}
Bartlett.Reflect.Plugin.Notifier.GrowlNotifier ..|> Bartlett.Reflect.Plugin.
@enduml
```

That may render something like



Chapter 9. Summary

Let’s review what we’ve learned about the command-line interface :

- It’s a Symfony Console Component [http://symfony.com/doc/current/components/console/index.html] that can be extended.
- We can produced Graphviz or PlantUML code syntax ready for image generation.

Part III. Processor Guide

Chapter 10. Architecture overview

Each processor is a specialized version of a graphical engine. UmlWriter version 1.0.0 is able to render only two diagram syntaxes :

- Graphviz [http://graphviz.org/]
- PlantUML [http://plantuml.sourceforge.net/]

But you are free to make your own. Here is the class diagram of processors architecture built with PlantUML processor

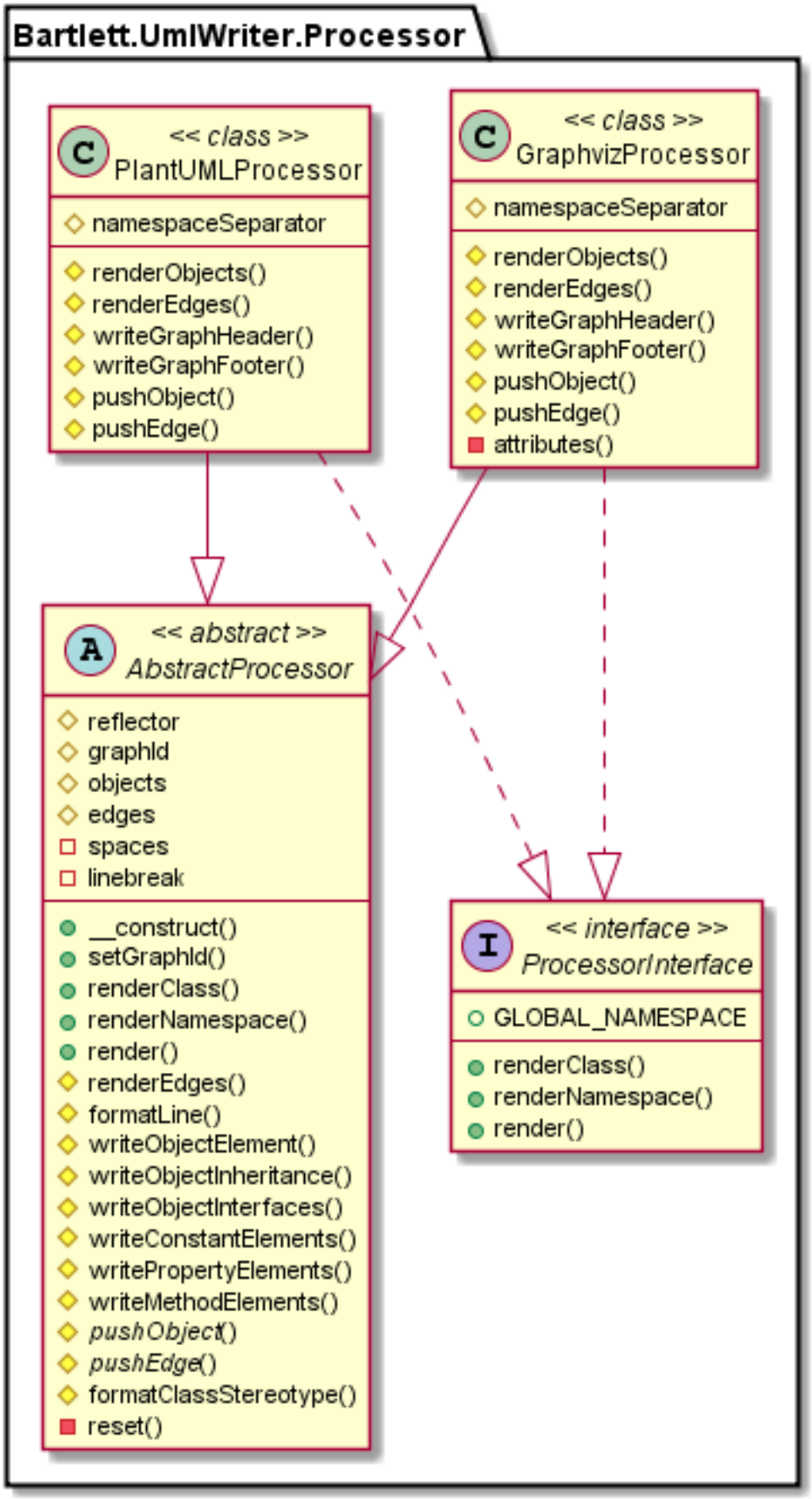
How To Build It. With the CLI mode, use the umlwriter command.

```
$ umlwriter diagram:render:namespace --processor=plantuml /path/to/umlwriter Ba
```

Output return the PlantUML diagram syntax, that you may redirect directly to a file

Then build a `png` image.

```
$ java -jar plantuml.jar -Tpng /path/to/puml_file -o /path/to/ouput/dir
```


Bartlett.UmlWriter.Processor
<< class >>
PlantUMLProcessor
namespaceSeparator
renderObjects()
renderEdges()
writeGraphHeader()
writeGraphFooter()
pushObject()
pushEdge()
<< class >>
GraphvizProcessor
namespaceSeparator
renderObjects()
renderEdges()
writeGraphHeader()
writeGraphFooter()
pushObject()
pushEdge()
attributes()
<< abstract >>
AbstractProcessor
reflector
graphId
objects
edges
spaces
linebreak
__construct()
setGraphId()
renderClass()
renderNamespace()
render()
renderEdges()
formatLine()
writeObjectElement()
writeObjectInheritance()
writeObjectInterfaces()
writeConstantElements()
writePropertyElements()
writeMethodElements()
pushObject()
pushEdge()
formatClassStereotype()
reset()
<< interface >>
ProcessorInterface
GLOBAL_NAMESPACE
renderClass()
renderNamespace()
render()

Any new processor must implement the `Bartlett\UmlWriter\Processor\ProcessorInterface` interface.

And to avoid to implement again and again the same methods, you should used the abstract `Bartlett\UmlWriter\Processor\AbstractProcessor` class, that will do the job for you.

There are three methods to implement :

- `renderClass()` that is in charge of return diagram syntax of a single class and its direct dependencies.
- `renderNamespace()` that is in charge of return diagram syntax of a single namespace and all its objects.
- `render()` that is in charge of return diagram syntax of all namespaces and objects.

Chapter 11. Methods

There are additional protected methods available in the abstract `Bartlett\UmlWriter\Processor\AbstractProcessor` class.

- `renderEdges()` that is in charge of building all links between objects (class, interface, trait)
- `formatLine()` that may format line with a left indentation.
- `writeObjectElement()` that render details of each object (class, interface, trait) in a namespace
- `writeObjectInheritance()` that render the class or interface parent.
- `writeObjectInterfaces()` that render all interfaces implemented by a class
- `writeConstantElements()`that render all constants of a class
- `writePropertyElements()` that render all properties of a class
- `writeMethodElements()` that render all methods of any objects (class, interface, trait)

There are two abstract methods you should implement :

- `pushObject()` to render an object with all details (attributes, operations)
- `pushEdge()` to render links (inheritances, implements) between objects

Other specialized methods that each processor must implement :

- `writeGraphHeader()` to render header of the main graph
- `writeGraphFooter()` to render footer of the main graph

Chapter 12. Limitations

The PlantUML processor is not yet able to print the PHP namespace \ (backslash), that is replaced by the **.** (dot)

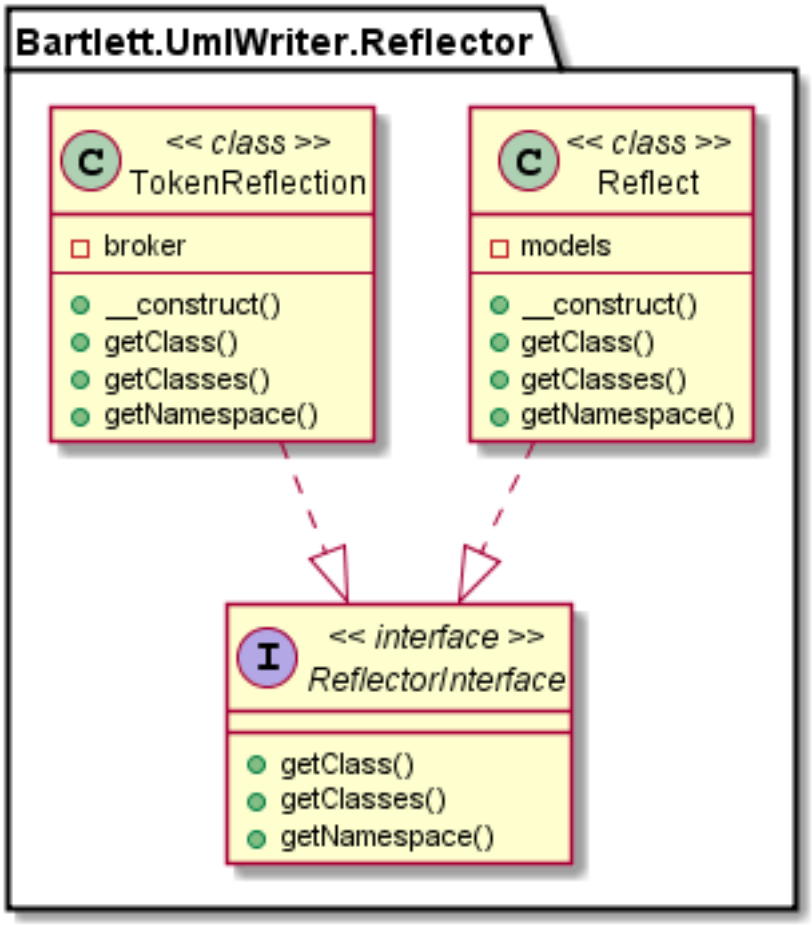
Part IV. Reflector Guide

Chapter 13. Architecture overview

Each reflector is a specialized version of a compatible reverse-engine. UmlWriter version 1.0.0 is able to use only two engines :

- Bartlett\Reflect [https://github.com/llaville/php-reflect] (default)
- Andrewsville\TokenReflection [https://github.com/Andrewsville/PHP-Token-Reflection]

Here is the class diagram of reflectors architecture (built with default PlantUML processor)



Any new reflector must implement the `Bartlett\UmlWriter\Reflector\ReflectorInterface` interface.

Chapter 14. Methods

- `getClass()` returns a compatible reflection class of the given object (FQN expected)
- `getClasses()` returns all objects from all namespaces
- `getNamespace()` returns a list of reflection class, of the given namespace