Chapter No. 5
"Using Code Documentation"
In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.5 "Using Code Documentation"

A synopsis of the book’s content

Information on where to buy this book

About the Author

M A Hossain Tonu graduated in Computer Science and Engineering from Dhaka University of Engineering and Technology (DUET) in Bangladesh. He has been a passionate developer over the past six years, has worked for leading software companies in the country, such as Somewherein and Improsys, and has developed a series of web applications, services, and solutions for foreign clients as part of the off shore software development and outsourcing team. He is an agile software craftsman, who loves to code, keep himself updated with cutting-edge technologies, and play with PHP, Zend Framework, Ruby-on-Rails, JavaScript, and more. He loves to moderate the local PHP community—phpXperts—and conducts seminars and workshops at different tech premises.

You can reach Tonu at mahtonu@gmail.com, and his tech blog is available at http://mahtonu.wordpress.com.

For More Information:
It takes many people to create a book like this, and I'd like to thank some people for their contributions to this work.

First of all, I would like to thank my wife Shamima Rahman Jhumur for her tremendous patience when I was unavailable to her. I appreciate my family, friends, and well-wishers, who continuously tolerate my computer madness.

To the people at Packt, I am much obliged: Kartikey Pandey, the man who started the process; Dayan Hyames, for guiding me throughout; Alka Nayak, the kind soul accepting my delays; and Lubna Shaikh, for being such a great help.

I'd also like to thank those who provided prepublication feedback, such as Ondřej Nešpor, Tomáš Myšík and Nurul Ferdous.

Lastly, I would like to dedicate this book to Hasin Hayder, the PHP mentor.
Productivity is an important factor for a software developer. A good development environment or surrounding tools with the essence of a particular programming flavor can boost up our coding productivity and yield a quality and optimized software product. In order to maintain a fast-paced development, developers seek the environment with which they feel at home. Such an Integrated Development Environment (IDE) can really accelerate code implementation and be the magic wand to your project development.

A good IDE is more like a Swiss army knife with crafted features. It consists of:

- A source editor
- A compiler/an interpreter
- A debugger
- Database management support
- Version Control System
- Tools for Object-Oriented Programming, such as Class Browser and Object Inspector

IDE, like NetBeans, comes with greater flexibility, with such features where the developer can feel at home. Moreover, NetBeans is absolutely free of charge and is provided by the open source community. Simply put, the IDE for PHP will facilitate your productivity from development to production, in every respect.

In this book, *PHP Application Development with NetBeans Beginner's Guide*, you will learn how to cover different categories of web-based applications with the help of NetBeans IDE through a couple of real-life, trendy PHP projects, and will complete the book as a confident PHP developer.

**What This Book Covers**

*Chapter 1, Setting Up Your Development Environment*, guides you through the process of NetBeans installation and sets up the PHP development environment step by step. By the end of this chapter, you will have your development environment ready on your operating system.

For More Information:

Chapter 2, Boosting Your Coding Productivity with PHP Editor, shows how you can write faster code using the NetBeans PHP Editor. You will be introduced to killer features of the IDE, such as code completion, code templates, rename refactoring, and code generation. At the end of this chapter, you will have a full, hands-on knowledge of the editor's smart features and increased coding productivity.

Chapter 3, Building a Facebook-like Status Poster using NetBeans, jumps directly to a real-life, PHP application development that will be used to display Facebook/Twitter-like, posted status streams. By the end of this chapter, you will be able to develop simple PHP applications with the NetBeans IDE.

Chapter 4, Debugging and Testing using NetBeans, will explain how to debug and test a PHP application using the IDE. Topics covered in the chapter include configuring XDebug, debugging the PHP source code, testing with PHPUnit and Selenium, and code coverage.

Chapter 5, Using Code Documentation, guides the developer through the process of creating source and project documentation. You will become familiar with PHPDoc standard tags and their use, to document the source code with the help of the editor. Also, you will use an external document generator for the project API.

Chapter 6, Understanding Git, the NetBeans Way, will show you how to use Git, a free and open source-distributed version control system. Using the IDE, you will be working on Git operations, such as initializing or cloning a repository, staging files, committing changes, reverting modifications, and remote repository operations such as fetching, pulling, and pushing, while working with branches. By the end of this chapter, you will be able to be part of a development team using the NetBeans collaborative development feature.

Chapter 7, Building User Registration, Login, and Logout, deals with a professional PHP application. You will design and develop a PHP application where users can register themselves, and after the registration they can log in to the application, view, and update their own profile, and more.

Appendix A, Introducing Symfony2 Support in NetBeans 7.2, will discover the Symfony2 PHP framework support by NetBeans. This introduces Symfony2's project creation, runs Symfony2 commands, and introduces bundle creation from NetBeans.

Appendix B, NetBeans Keyboard Shortcuts, is a convenient reference for common NetBeans keyboard shortcuts.

In this chapter, we are going to document our PHP source code using the NetBeans IDE. We will learn how to quickly document variables, methods, classes, or the entire project, and discuss the following issues:

- Convention for source documentation
- How to document the source code
- PHP project API documentation

**Writing great documentation**

Coding is the art of instructing machines, and when it comes to human readability, code should be expressive, self-explanatory, and beautiful. The code should be reusable and understandable, so that you can use it again a couple of months from now. A good practitioner makes the code as simple as possible to understand, and keeps the code documentation where it is really required.

Code documentation is the motivating part of coding, particularly when you are working in a collaborative team environment; documentation should be done in a sensible way, so that learning the intent of the code can be faster among collaborators.

A regular practice to document the source code is putting comments in your code specified by the **phpDoc** format, so that your code becomes more meaningful and the external documentation generator can parse such comments.

For More Information:
PHPDoc – commenting standard for PHP

PHPDoc is an adaptation of Javadoc for the PHP programming language. Since it is the standard for commenting the PHP code, it allows external document generators, such as phpDocumentor and ApiGen to generate HTML documentation for APIs. It helps a variety of IDEs, such as NetBeans, PhpStorm, Zend Studio, and Aptana Studio, to interpret variable types and provide improved code completion, type hinting, and debugging. According to PHPDoc, documentation is written using text blocks named DocBlock, which precede the element to be documented. As a way of describing programming constructs such as class, interface, functions, methods, and so on, tag annotations are used inside the DocBlock.

Example of a DocBlock

A DocBlock is an extended C++ style PHP comment that begins with "/**" and has "*/" at the beginning of every line.

    /**
     * This is a DocBlock comment
     */

A DocBlock contains three basic segments, in this order:

- Short description
- Long description
- Tags

Example:

    /**
     * Short description
     *
     * Long description first sentence starts here
     * and continues on this line for a while
     * finally concluding here at the end of
     * this paragraph
     *
     * The blank line above denotes a paragraph break
     */

For More Information:

The short description starts on the first line, and can be terminated with a blank line or a period. A period inside a word (for example example.com or 0.1 %) is ignored. If the short description becomes more than three lines long, then only the first line is taken. The long description continues for as many lines as required, and may contain the HTML markup for display formatting. The external document parser will convert all the whitespaces into a single space in the long description, and may use paragraph breaks to define newlines, or <pre>, as discussed in the following section.

The long and short description of a DocBlock is parsed for a few selected HTML tags, which determine additional formatting using the followings tags:

- <b>: This tag is used to emphasize/bold the text
- <code>: This tag is used to surround the PHP code; some converters will highlight it
- <br>: This tag is used to provide a hard-line break, and may be ignored by some converters
- <i>: This tag is used to italicize/mark the text as important
- <kbd>: This tag is used to denote the keyboard input/screen display
- <li>: This tag is used to list items
- <ol>: This tag is used to create an ordered list
- <ul>: This tag is used to create an unordered list
- <p>: This tag is used to enclose all the paragraphs; otherwise, content will be considered text
- <pre>: This tag is used to preserve line breaks and spacing, and assumes all tags are text (like XML's CDATA)
- <samp>: This tag is used to denote samples or examples (non-PHP)
- <var>: This tag is used to denote a variable name

In rare cases when the text "<b>" is needed in a DocBlock, use a double delimiter, as in <<b>>. The external document generator will automatically translate that to the physical text "<b>".

**Acquainting with PHPDoc tags**

PHPDoc tags are single words prefixed by an @ symbol, and are only parsed if they are the first thing on the new line of a DocBlock. A DocBlock precedes before structural elements, and such elements can be programming constructs, such as namespaces, classes, interfaces, traits, functions, methods, properties, constants, and variables.

---

**For More Information:**

Some common lists of tags with details have been divided into groups for better understanding, as follows:

**Data type tags**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@param</td>
<td>Type [$varname] description</td>
<td>Documents a function or method parameter.</td>
</tr>
<tr>
<td>@return</td>
<td>Type description</td>
<td>Documents the return type of a function or method. This tag should not be used for constructors or methods defined with a void return type.</td>
</tr>
<tr>
<td>@var</td>
<td>Type</td>
<td>Documents the data type for a class variable or constant.</td>
</tr>
</tbody>
</table>

**Legal tags**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@author</td>
<td>Author name <a href="mailto:author@email">author@email</a></td>
<td>Documents the author of the current element</td>
</tr>
<tr>
<td>@copyright</td>
<td>Name date</td>
<td>Documents copyright information</td>
</tr>
<tr>
<td>@license</td>
<td>URL name</td>
<td>Is used to indicate which license is applicable for the associated structural elements</td>
</tr>
</tbody>
</table>

**Versioning tags**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@version</td>
<td>Version string</td>
<td>Provides the version number of a class or method</td>
</tr>
<tr>
<td>@since</td>
<td>Version string</td>
<td>Documents the release version</td>
</tr>
<tr>
<td>@deprecated</td>
<td>Version description</td>
<td>Is used to indicate which elements are deprecated and are to be removed in a future version</td>
</tr>
<tr>
<td>@todo</td>
<td>Information string</td>
<td>Documents things that need to be done to the code at a later date</td>
</tr>
</tbody>
</table>

For More Information:

Other tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Usage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@example</td>
<td>@path/to/example</td>
<td>Documents the location of an external saved example file</td>
</tr>
<tr>
<td>@link</td>
<td>URL link text</td>
<td>Documents the URL reference</td>
</tr>
<tr>
<td>@see</td>
<td>Element name(s) separated by comma</td>
<td>Documents any element</td>
</tr>
<tr>
<td>@uses</td>
<td>Name of element</td>
<td>Documents how the element is used</td>
</tr>
<tr>
<td>@package</td>
<td>Name of a package</td>
<td>Documents a group of related classes and functions</td>
</tr>
<tr>
<td>@subpackage</td>
<td>Name of sub package</td>
<td>Documents a group of related classes and functions</td>
</tr>
</tbody>
</table>

Among the most used tags, @param and @return can be used only for functions and methods, @var for properties and constants, @package and @subpackage for procedural pages or classes, while other tags, such as @author, @version, and so on, can be used for any element. Besides these tags, @example and @link can be used as inline tags.

---

You can find the list of tags at [http://www.phpdoc.org/docs/latest/for-users/list-of-tags.html](http://www.phpdoc.org/docs/latest/for-users/list-of-tags.html).

Now, we will dive into documenting our PHP source code using NetBeans.

Documenting the source code

In this section, we will learn to document functions, methods, classes, interfaces, global variables, constants, and so on, and discuss the benefits of using such a code documentation. As discussed earlier, in the collaborative development environment, the description of methods, classes, and so on, are very important to learn the intent of the code, and we will see that implemented practically in this section.

Now, create a new PHP project named Chapter5 in NetBeans, and use it for all the following tutorials.

Documenting the functions and methods

In this section, we will learn to use the NetBeans auto documentation feature at the beginning of a PHP function or method.

For More Information:

Time for action – documenting a PHP function or method

In this tutorial, let's create a simple PHP function or method with some parameters passed into it and different types of variables declared inside it. We are just practicing to see how the NetBeans auto documentation generator works on such commonly used structural elements. Let's go through the following steps:

1. Add a PHP file named sample1.php into the project, and type a PHP function as follows:

```php
function testFunc(DateTime $param1, $param2, string $param3 = NULL)
{
    $number = 7;

    return $number;
}
```

In this function we can see that there are three parameters passed into the `testFunc` method—`$param1` as `DateTime`, `$param2` without type-hinting as it may have a mixed type of value, and `$param3` is optional with a default `NULL` value. Also, inside the body, the function contains one integer type variable and returns that integer type as well.

2. Type /** in the line before the `testFunc` function, and press Enter. You can see that NetBeans parses the function and generates the documentation before the function according to the PHPDoc standard, which looks similar to the following:

```php
/**
 * @param DateTime $param1
 * @param type $param2
 * @param string $param3
 * @return int
 */
function testFunc(DateTime $param1, $param2, string $param3 = NULL)
{
    $number = 7;

    return $number;
}
```

For More Information:
In the previous code snippet, we can see that NetBeans generated the documentation mentioning the parameters and return type, which are listed as follows:

- Parameters are annotated with the `@param` tag and parameter type from the given type hinting
- The return type is annotated with `@return`

You can see that the type and name beside each tag are separated with a whitespace. In case the type hint is unavailable, then NetBeans leaves it as a simple type, such as `$param2`. The word that is usually used in documentation is "mixed" when the real data type is unknown and you can edit that "type" as well.

3. You can add a description of each variable in the documentation; beside the variable name, just put the description with a leading whitespace as follows:

```php
/**
 * @param DateTime $param1 this is parameter1
 * @param array $param2 this is parameter2
 * @param string $param3 this is parameter3 which is optional
 * @return int what is returned, goes here
 */
```

4. Also, you may want to add a short description for the documentation, which looks similar to the following:

```php
/**
 * a short description goes here
 *
 * @param DateTime $param1 this is parameter1
 * @param array $param2 this is parameter2
 * @param string $param3 this is parameter3 which is optional
 * @return int what is returned, goes here
 */
```

For More Information:

5. Now, let’s see how this NetBeans-generated documentation looks, while someone tries to call this `testFunc` from anywhere within the project. Try to type the function name anywhere. Say, let’s start typing the function name in the `index.php` file inside the project, and you will see the NetBeans auto-suggestion of that function name with parameter hints and the documentation, as follows:

![NetBeans auto-documentation screenshot](image)

If documentation is available for a function or any element, then NetBeans shows the documentation while auto-suggestion is in process, as shown in the previous screenshot.

**What just happened?**

We just learned how to use the NetBeans auto documentation generator. By typing `/**` and pressing `Enter` before the functions, we can parse the metadata and generate the documentation. We can update the documentation as well. Again, the external document generator can extract such DocBlocks to create the project API documentation. Now, we will add a documentation before a PHP class in the next section.

---

**For More Information:**

Documenting classes

A document before a class is very important to learn about the class and its usage. A best practice is to decorate the preceding documentation with proper annotations, such as `@package`, `@author`, `@copyright`, `@license`, `@link`, and `@version`, and with a proper description of the class.

Time for action – documenting the PHP class and class variables

In this section, we will add a PHP class using NetBeans and update the preceding DocBlock with class documentation tags. So let’s go for it...

1. Right-click on the `Chapter5` project to choose **New | PHP Class...**, insert the class name as `Test` into the **File Name** box, and click on **Finish**, as follows:

![New PHP Class dialog box](image)

For More Information:

2. The Test class should look similar to the following:

```
<?php

/**
 * Description of Test
 */

class Test {
    //put your code here
}
?>
```

In the previous screenshot, you can see that the opened Test class has an added DocBlock at the top with a sample class description along with the @author tag.

3. You may want to add PHPDoc tags before the line containing the @author tag; say you want to add the @package tag as soon as you type @p. The NetBeans code auto completion feature shows tags starting with @p to have a description that looks similar to the following screenshot:

```
<?php

/**
 * Description of Test
 */

@package ApiGen

Specify package to group classes or functions and defines into.

@package packagename

Description

@package can only be used to document procedural pages or classes.

Packages are used to help you logically group related elements. You write classes to group related functions and data together, and \$docDocumentor represents the contents of files (functions, defines, and includes) as "Procedural Pages." A package is used to group classes and procedural pages together in the same manner that a directory groups related files together.
```

For More Information:
4. Update the DocBlock with your own, so it looks similar to the following:
/**
 * Short description of the Test Class
 * Long multiline description of the Test Class goes here
 * Note: any notes required
 * @package Chapter5
 * @author M A Hossain Tonu
 * @version 1.0
 * @copyright never
 * @link http://mahtonu.wordpress.com
 */

5. In the above documentation, you can see that corresponding tags have been added for the class, so that the class information is available while you try to instantiate the class object with code completion, as follows:

Also, such a class DocBlock can be extracted using the external API documentation generator.

For More Information:
6. Now, type a class variable named $variable into the Test class as follows:
   public $variable;

7. To add the class variable documentation, type /**, and hit Enter before the line in
   which it is declared, so that the documentation looks similar to the following:
   /**
    * @var type
   */

8. Here, you may update the block as follows:
   /**
    * example of documenting a variable's type
    * @var string
   */

9. In order to view the class hierarchy tree, at a later section, you may add a child class
   in our project named TestChild extending the Test class, which looks similar to the following:
   /**
    * Short description of the TestChild Class
    *
    * Long multiline description of the TestChild Class goes here
    *
    * Note: any notes required
    * @package Chapter5
    * @author M A Hossain Tonu
    * @version 1.0
    * @copyright never
    * @link http://mahtonu.wordpress.com
    */
   class TestChild extends Test {
   }

For More Information:
What just happened?
We have practiced how to add documentation before PHP functions, classes, and its attributes using the PHPDoc format, and tested how this documented information becomes available throughout the project. The same style of DocBlock or appropriate tags can be applicable to document the PHP interfaces as well.

Documenting TODO tasks
You can use the @todo tag for an element, to document planned changes, which have not been implemented yet, and the tag may be used for almost any element that can be documented (global variable, constant, function, method, define, class, and variable).

Time for action – using @todo tags
In this tutorial, we will learn to use @todo tags to document our future tasks, and will view the tasks list from the NetBeans task or action items window:

1. Inside the TestChild PHP class or in the preceding document block of the class, we can use a @todo tag; inside multiple line comments or DocBlock, add a tag similar to the following:

   /**
    * @todo have to add class variable and functions
    */

   In the above document block, we can see that the task has been described beside the tag separated by a whitespace. Also, a @todo tag can be added using a single comment line as follows:

   //TODO need to add class variable and functions

2. So, the TestChild class may look similar to the following:

   class TestChild extends Test {

   //TODO have to add class variable and functions
   

For More Information:
3. As we add the tasks inside the file, the task should be visible in NetBeans' **Tasks** or **Action Items** window; press *Ctrl + 6* to open the window, and the added task should be listed in the **Tasks** window, as shown in the following screenshot:

![Screenshot of NetBeans with Tasks window](image)

**What just happened?**

NetBeans updates the task list in the **Task** window as soon as a new task is added using the **TODO** task marker, and you can have all the tasks listed in that window for the entire project or for all the opened projects at NetBeans. Such tags can be used when we have something in mind that we want to implement and haven’t got enough time to write the code, considering its future implementation. So, you can drop the idea in appropriate places using the **@todo** tag.

For More Information:

So far, we have learned how to use PHPDoc standard tags to document the PHP source elements, and also dealt with DocBlock to write source documentation. Elementary concepts regarding source documentation have been discussed. So, in our next section, we will learn to extract such DocBlocks, to generate HTML documentation for the entire project or API.

**Documenting the API**

As we have already discussed the significance of source documentation, the documentation should be presented to a general user in a well-organized way or graphically elaborated using HTML pages. Such API documentation, converted from source DocBlocks, can be the technical documentation used for learning about the source. NetBeans comes with the support to generate API documentation from the PHP source for the entire project using the **ApiGen** auto documentor tool.

ApiGen is the tool for creating API documentation using the PHPDoc standard, and supports the latest PHP 5.3 features, such as namespaces, packages, linking between documentation, cross referencing to PHP standard classes and general documentation, creation of highlighted source code, and support for PHP 5.4 traits. It also generates a page with trees of classes, interfaces, traits, and exceptions for the project.

Check out the ApiGen features at [http://apigen.org/##features](http://apigen.org/##features).

In the next section, we will discuss how to install ApiGen and configure it with NetBeans.

**Configuring ApiGen**

We will first install ApiGen via PEAR and configure it with NetBeans, so that we can generate the API documentation from the IDE. We can automatically install ApiGen and all of its dependencies with PEAR auto-discover enabled. Enabling the discover feature not only automatically adds ApiGen to the system PATH, but also allows for an easy updating of each ApiGen component.

For More Information:
**Time for action – installing ApiGen and configuring it with NetBeans**

We are already familiar with installing PHP libraries via PEAR (discussed in the previous chapter), and we may have had the PEAR configuration `auto_discover` set to ON. In this section, we will install and configure ApiGen with NetBeans, using the following steps:

1. From the terminal or command prompt, run the following commands to install ApiGen:
   ```
   pear config-set auto_discover 1
   pear install pear.apigen.org/apigen
   ```
   The `install` command will automatically download and install ApiGen along with all of its dependencies. Skip the first command if you have already enabled PEAR `auto_discover`.

2. Now, we need to add the ApiGen executable file to the IDE. Open the IDE Options window from **Tools | Options**, and choose the **PHP Tab | ApiGen** tab, then click on the **Search...** button to search for ApiGen scripts. The ApiGen scripts should be listed automatically, as shown in the following screenshot:

   ![ApiGen scripts](image)

   ![Image](image)

For More Information:
3. From the previous screenshot, select `apigen.bat` for Windows OS or `apigen` for other OSes, and press **OK** to have the ApiGen script integrated with the IDE, as shown in the following screenshot:

![ApiGen Script Integration](image)

You may browse the ApiGen script path there as well.

4. Press **OK** to save the settings.

**What just happened?**

Up until now, we have configured the ApiGen tool with NetBeans, and this tool is ready to be used for PHP projects. Once you have integrated the tool with the IDE, you may want to use it from the IDE for the HTML documentation generation for your PHP projects. In our next tutorial, we will learn the usage of the tool from the IDE.

**Generating API documentation**

We will generate the HTML documentation using ApiGen for the sample PHP project `Chapter5`, and the tool extracts documentation from the DocBlocks available in the project. The generation process can be viewable in the **Output** window of the IDE. Finally, the generated HTML documentation opens in the web browser.

---

**For More Information:**

Time for action – generating documentation using ApiGen

Using integrated ApiGen from the IDE, we will run the documentation generator. Note that we need to define the target directory to store the HTML documents. Let’s create the HTML documentation for our sample project, according to the following steps:

1. Right-click on the chapter5 project node. From the context menu, choose Properties | ApiGen, and the following Project Properties window will be displayed:

2. From the previous Project Properties window, define the Target Directory where the HTML pages will be stored, and uncheck the PHP box for excluding PHP’s default elements from the documentation. In this project, let’s create a directory named doc as the target directory inside the project, so that the documentation can be browsed at http://localhost/chapter5/doc/.

3. Click on OK to save the settings.

For More Information:
4. Now, right-click on the `chapter5` project node. This will generate a menu, which will look similar to the following screenshot:

![Project Menu Screenshot](image)

5. From the previous project context menu, choose **Generate Documentation** to start the HTML document generation process, from the given DocBlocks.

6. As soon as we chose **Generate Documentation** in the previous step, the HTML documentation generator started to progress, and completed the HTML documentation. The generation process is summarized in the **Output** window, as follows:

![Output Window Screenshot](image)

For More Information:

7. Also the HTML documentation for the entire project has been opened in a browser that looks similar to the following:

In the above screenshot, we can see that the HTML documentation has been created for the entire project. The documentation is organized according to packages, classes, and functions as per the left frame.

8. Browse through the links created for the project, and explore how the classes and methods are represented over there. You may click on the TestChild class link in the previous window to get the following screenshot:

For More Information:
9. In the above screenshot, we can see that the class inheritance is also represented using a tree along with a suitably decorated documentation for the class, as per its DocBlock.

**What just happened?**

We created a professional API documentation from the source code comments block, and discovered how the classes were organized properly in the final documentation. Note that ApiGen provides a searching facility for classes, functions, and so on at the generated HTML interface, and also provides a customizable template facility to modify the overall documentation’s look. We are now confident enough to document the PHP source code effectively.
Pop quiz – reviewing tags

1. Which of the following tags is applicable for only functions or methods?
   a. @author
   b. @package
   c. @param
   d. @link

2. Which of the following tags can be used to document the release version of any element?
   a. @version
   b. @since
   c. @deprecated
   d. @todo

3. Which of the following tags can be used as an inline tag?
   a. @example
   b. @param
   c. @version
   d. @see

Have a go hero – doing more with documentation

Each time you run the NetBeans documentation generator, it wipes out the target directory and creates a new set of HTML documents there. Try commenting interfaces, constants, traits, and so on, and run the documentation generator to test the generated API documents.

For More Information:
Summary

In this chapter, we have discussed and practiced how to document the source code for PHP applications using NetBeans.

We have specially focused on the following topics:

- PHPDoc standards and tags
- Documenting PHP functions/methods, classes, and its variables
- Documenting TODO tasks
- Configuring ApiGen with NetBeans
- API documentation using ApiGen

Finally, it was fun to use the auto-doc generator, and get the HTML documentation generated within a few seconds.

As we are going for the collaborative PHP development in our next chapter, such source documentation is required, in order to maintain a good practice within the development team. In the next chapter, we will learn to use the version control system (Git) from NetBeans.

For More Information:

Where to buy this book


Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

For More Information: