

Tomasz "Zyx" Jędrzejewski

Dokumentowanie kodu PHP w phpDocumentor

Wersja 2.0 (24.05.2009)



Szczegółowe informacje o licencji znajdują się pod artykułem.

www.zyxist.com

Dokumentowanie kodu PHP w phpDocumentor

Bez względu na to czy piszemy kod na własne potrzeby czy też zamierzamy go publikować, powinien być on właściwie udokumentowany. Dokumentacja przyda się zarówno końcowym użytkownikom, jak i nam, gdyż będziemy mieć listę wraz z opisem wszystkich klas, metod i funkcji, jakie stosujemy w naszym kodzie, a w przypadku większych projektów ich ilość może iść w setki i tysiące. Oczywiście jest, że zapamiętanie ich wszystkich nie wchodzi w grę, podobnie jak szukanie potrzebnej funkcji w dziesiątkach tysięcy linii kodu, by sprawdzić, co należy podawać jako drugi argument.

W praktyce możemy wyróżnić dwa rodzaje dokumentacji. Pierwsza z nich to dokumentacja samego API naszego skryptu, czyli zestawienie wszystkich funkcji i klas. Druga to dokumentacja użytkownika, która skupia się bardziej na wyjaśnieniu zasady działania i korzystania, pełni zatem rolę zbioru praktycznych poradników dla końcowych użytkowników produktu. Przykład takiego podziału możemy znaleźć np. w [Zend Frameworku](#). Na stronie projektu w sekcji „Documentation” znajdziemy dwa odnośniki:

1. *APIs* – dokumentacja całego API frameworka, gdzie możemy odnaleźć szybko konkretną funkcję, dowiedzieć się, co ona robi i jakie ma argumenty.
2. *Reference Guide* – zbiór artykułów-poradników do każdego komponentu frameworka, które wyjaśniają filozofię jego działania oraz na przykładach pokazują, jak z niego korzystać.

Choć właściwie w każdym narzędziu dokumentującym możemy zrealizować jeden i drugi cel, najczęściej specjalizuje się ono w jednym z nich, a reszta traktowana jest jako dodatek funkcjonalny. W tym artykule pragnę przyjrzeć się bliżej aplikacji *phpDocumentor*, która służy do generowania dokumentacji API bezpośrednio z kodu źródłowego naszego projektu PHP.

Zasada działania

phpDocumentor buduje dokumentację na podstawie kodu źródłowego naszego projektu. Bazuje on na znajomości języka PHP, dzięki czemu potrafi odnaleźć w nim klasy, funkcje i metody, a następnie każdą z nich prawidłowo opisać. Ponadto, nad każdym elementem mamy możliwość dodania specjalnego komentarza z dodatkowym opisem. Aplikacja parsuje jego treść i umieszcza w dokumentacji, dodatkowo wyszukując w nim specjalne tagi. Dzięki nim programista może opisać przeznaczenie argumentów funkcji czy zwracaną wartość. Taki styl dokumentowania określany jest często mianem *phpdoc*, a bardzo podobne w działaniu i składni wynalazki można znaleźć również do wielu innych języków programowania (np. Java, D, Python...).

Instalacja

W chwili pisania tego tekstu, najnowszą dostępną wersją jest *phpDocumentor* 1.4.2. Działa on prawidłowo na PHP5 i można go ściągnąć manualnie albo z repozytorium PEAR. Pokażemy tutaj oba rodzaje instalacji. Aby zainstalować aplikację ręcznie, należy odwiedzić stronę www.phpdoc.org i ściągnąć najnowszy dostępny pakiet z sekcji „Download”. Ściągnięte archiwum będzie zawierać plik *package.xml* dla PEAR oraz dodatkowy podkatalog, którego zawartość kopiujemy gdzieś, gdzie będzie ona dostępna z poziomu serwera WWW. Przykładowo, u mnie *localhost* podpięty jest pod katalog */home/httpd/*, dlatego zawartość skopiowałem do */home/httpd/phpdoc*, dzięki czemu mogę dostać się do skryptu poprzez adres <http://localhost/phpdoc>. Jest to właściwie wszystko, niemniej w skład aplikacji wchodzi także interfejs linii komend. Aby z niego skorzystać, należy wyedytować zmienną środowiskową *PATH*, dodając do niej ścieżkę do *phpDocumentora*. W systemie Windows wartość tej zmiennej możesz skonfigurować w Panelu Sterowania w aplikacji „System”. Użytkownikom uniksowym raczej nie trzeba tłumaczyć, gdzie jej szukać.

Instalację z repozytorium PEAR możemy wykonać z wiersza poleceń systemu operacyjnego pod warunkiem, że zmienna *PATH* jest prawidłowo ustawiona. Uruchamiamy konsolę i wpisujemy:

```
pear install PhpDocumentor
```

PEAR automatycznie odnajdzie potrzebny pakiet i zainstaluje go w domyślnej lokalizacji repozytorium. Zaletą tego rozwiązania jest prostota oraz fakt, że najczęściej interfejs linii komend będzie od razu gotowy do uruchomienia. Dostępność interfejsu WWW zależy od tego, jak skonfigurowaliśmy nasz serwer.

Pierwsze użycie

Aby sprawdzić czy wszystko działa, możemy uruchomić phpDocumentora. Graficzny konfigurator dostępny będzie poprzez nasz serwer WWW – w moim przypadku <http://localhost/phpdoc>, natomiast u Ciebie zależy to od tego, gdzie zainstalowałeś skrypt lub jaką masz konfigurację. Ekran podzielony jest na dwie części. W górnej znajdują się formularze konfiguracyjne, zaś w dolnej komunikaty systemu dotyczące np. postępów w generowaniu dokumentacji.

Interfejs linii komend możemy uruchomić bardzo prosto, wystarczy w konsoli wpisać:

```
phpdoc
```

Ponieważ nie wymieniliśmy żadnych opcji, domyślnie wyświetli nam się krótka informacja o programie oraz pomoc. Spróbujmy wygenerować dokumentację do jednego z już posiadanych projektów. PhpDocumentor jest to w stanie zrobić, ponieważ część informacji można wywnioskować bezpośrednio z kodu, choć oczywiście żaden element nie będzie posiadać dołączonych opisów. W interfejsie WWW wybierz zakładkę „Files” i w polu „Directory to parse” podaj ścieżkę lub listę ścieżek oddzielonych przecinkami do katalogów, z których należy utworzyć dokumentację. W „Output” podaj ścieżkę do katalogu, gdzie ma być zapisany wynik oraz wybierz styl graficzny. Domyślnie dostępnych jest kilka stylów HTML-owych, PDF, CHM oraz DocBook. Zajrzyj jeszcze do zakładki „Options”, gdzie możesz np. nadać tytuł.

W przypadku korzystania z wiersza poleceń, musimy wszystkie te dane wpisać jako argumenty wywołania:

```
phpdoc -o HTML:frames:earthli -d /home/zyxist/Projekty/mojprojekt -t /home/zyxist/Temp/phpdoc -ti „Mój projekt”
```

Opis poszczególnych opcji:

1. `-o` – wybór systemu wyjścia.
2. `-d` – katalog lub lista katalogów do przetworzenia
3. `-t` – katalog na wynik.
4. `-ti` – tytuł dokumentacji (domyślnie „Generated documentation”).

Jeśli wszystko poszło dobrze, w obu przypadkach powinieneś ujrzeć listę komunikatów informujących o postępach generowania i po kilkunastu sekundach w folderze wskazywanym przez „Target” dostaniesz gotową dokumentację w żądanym formacie.

Dokumentowanie kodu w komentarzach

Po bliższym zapoznaniu się z wygenerowaną dokumentacją widać, że poszczególne funkcje, klasy i metody nie mają żadnych opisów, więc dla czytelnika ich zastosowanie może być trudne do odgadnięcia. Dlatego musimy teraz dodać do kodu PHP specjalne komentarze dokumentacyjne, w których dołączymy wszystkie niezbędne elementy. Poniżej widzimy przykładową klasę, na przykładzie której pokażemy, jak to wszystko działa:

```
<?php
class mojaKlasa
{
    public $pole = 'wartosc';
    private $_polePrywatne = null;

    public function metoda($a, $b, $c = null)
    {
        /* tu jakas tresc */
    } // end metoda();
} // end mojaKlasa;
```

Komentarz dokumentacyjny wygląda następująco:

```
/**
 * to jest komentarz dokumentacyjny
```

```
* to jest komentarz dokumentacyjny
*/
```

Ma on zawsze postać komentarza wielolinijkowego, lecz rozpoczyna się dwoma gwiazdkami, zamiast jednej. Ponadto każdą kolejną linię również musimy rozpoczynać od gwiazdki w dokładnie taki sam sposób, jak pokazane jest w przykładzie, z zachowaniem niezbędnych wcięć. Utrzymanie dyscypliny przy tak ścisłym rygorze może wydawać się trudne, lecz na szczęście każdy porządny edytor kodu będzie posiadać wbudowane wsparcie dla tego typu komentarzy. W pakiecie NetBeans moduł PHP natychmiast rozpoznaje komentarz dokumentacyjny i sam dba o wykonanie niezbędnych wcięć oraz poprzedzenie każdej linijki gwiazdką, a ponadto podpowiada nam, jakie elementy możemy w takim komentarzu wykorzystać.

Ważną zaletą komentarzy dokumentacyjnych jest to, iż duże pakiety programistyczne (np. Eclipse czy NetBeans) potrafią na ich podstawie na bieżąco wyświetlać opisy do funkcji, z których właśnie chcemy korzystać. Wpisując *mojaFunkcja()*, natychmiast lokalizują one komentarz dokumentacyjny i wyświetlają jego treść w postaci ramki informacyjnej tak, byśmy od razu mogli zapoznać się z wymaganymi argumentami oraz jej opisem.

Komentarz dokumentacyjny umieszczamy zawsze bezpośrednio nad elementem, który opisujemy. W naszym przypadku będzie wyglądać to następująco:

```
<?php
/**
 * Komentarz do pliku
 */

/**
 * Komentarz do klasy
 */
class mojaKlasa
{
    /**
     * Komentarz do pola.
     */
    public $pole = 'wartosc';
    /**
     * Komentarz do pola.
     */
    private $_polePrywatne = null;

    /**
     * Komentarz do metody
     */
    public function metoda($a, $b, $c = null)
    {
        /* tu jakas tresc */
    } // end metoda();
} // end mojaKlasa;
```

Przyjrzyjmy się teraz dokładniej budowie treści komentarza. Póki co umieściliśmy tam jedynie najprostszy opis, jednak możliwości systemu są dużo większe. Na początek podzielmy opis na streszczenie i dokładne objaśnienie:

```
/**
 * To jest streszczenie działania funkcji.
 *
 * Po jednej linijce przerwy zamieszczamy
 * dokładniejszy, wielolinijkowy opis.
 */
```

Aby dodać jakieś formatowanie, możemy użyć zwykłego HTML-a.

Tagi i pakiety

Nasz skrypt bardzo często podzielony jest na mniejsze części, tworzące pewną logiczną całość.

Przysłowiowe wrzucenie wszystkiego do jednego worka nie wyszłoby na dobre czytelności takiej dokumentacji, gdyż wszystkie klasy byłyby wymieszane ze sobą i nie dałoby się uchwycić powiązań między nimi. Aby poradzić sobie z tym problemem, nie musimy generować dla każdego modułu osobnej dokumentacji. Zamiast tego wystarczy pogrupować poszczególne klasy w pakiety. Przynależność do pakietu można określić za pomocą tzw. tagów:

```
/**
 * To jest opis pewnej klasy.
 *
 * @package pakiet1
 */
```

Znacznik *@package* służy właśnie do zaznaczania pakietu, do którego dany element należy. Jako argument podajemy nazwę pakietu. PhpDocumentor automatycznie wygeneruje tyle podstron pakietów, ile użyliśmy w kodzie źródłowym, zatem ich istnienia nie trzeba nigdzie dodatkowo potwierdzać. Dostępnych tagów jest dużo więcej, a poniżej zamieszczam listę tych najprzydatniejszych:

```
/**
 * To jest opis pewnej klasy.
 *
 * @access public
 * @author Jan Kowalski <adres@email.com>
 * @license http://www.example.com/license/licencja
 * @version 1.0
 * @copyright Jan Kowalski 2009
 * @internal
 * @param string $argument1 Opis argumentu
 * @return array
 * @var string
 * @static
 * @final
 * @abstract
 * @todo implement this and that
 * @deprecated deprecated since version 0.9
 */
```

Znaczenie niektórych tagów:

- *@access* – dostęp do elementu (prywatny, publiczny itd.)
- *@internal* – element stanowi wewnętrzną część systemu, o której końcowy użytkownik nie musi wiedzieć. Możemy poinformować phpDocumentora, aby ignorował elementy oznaczone tym tagiem, tworząc wersję dla użytkownika końcowego oraz dla twórców aplikacji.
- *@param* – tego tagu można używać wielokrotnie do opisu poszczególnych argumentów funkcji itd. Ważne jest, aby dobrze określić typ, gdyż dobre pakiety programistyczne potrafią na podstawie tego podpowiadać listę dostępnych metod w klasach, śledząc typy zmiennych. Nazwy zmiennych muszą pokrywać się z tymi, których faktycznie użyliśmy w deklaracji metody/funkcji.
- *@return* – typ zwracany przez funkcję. Również zalecane jest dokładne jego określenie z tych samych powodów.
- *@var* – typ zmiennej wewnętrznej (pola) klasy, który także powinniśmy poprawnie ustawić.
- *@static* – oznaczenie elementu jako statycznego.
- *@final* – oznaczenie elementu jako finalnego.
- *@abstract* – klasa lub metoda abstrakcyjna.

Kompilacja

Podawanie wszystkich argumentów generowanej dokumentacji przez linię komend lub interfejs WWW nie jest zbyt wygodne, gdy chcemy wielokrotnie generować tę samą, ale zaktualizowaną dokumentację. Proces ten można jednak zautomatyzować, tworząc plik konfiguracyjny. Jego lokalizacja jest dowolna, a on sam korzysta ze składni typowej dla plików INI. Przykładowy plik umieszczam poniżej:

```
[Parse Data]
;; Jeśli czegoś nie chcesz ustawiać, poprzedź to średnikiem, by nie było
;; interpretowane przez parser.

;; tytuł dokumentacji
title = Moja dokumentacja

;; Czy parsować pliki zaczynające się od kropki, np. .bash_profile, .htaccess
;; wartości: true, false
hidden = false

;; Czy pokazywać elementy prywatne (@access private oraz @internal)?
parseprivate = on

;; Twoje własne tagi
;; wartości: jakiegokolwiek nazwy oddzielone przecinkami (bez mały)
customtags = call

;; Jaki jest główny pakiet projektu?
;; akceptowane wartości: liczby, cyfry, podkreślenie (_) i pauza (-).
defaultpackagename = pakiet1

;; Czy NIE wyświetlać informacje o postępie kompilacji?
;; Wartości: on, off
quiet = off

;; Gdzie powinna być zapisana gotowa dokumentacja?
target = /home/uzytkownik/Dokumentacje/moja
;; Pakiety, które MUSZA być włączone. Jeśli nie ustawisz tego pola,
;; phpDocumentor uwzględni wszystkie pakiety z kodu
;; źródłowego
packageoutput = package1,package2

;; Pliki do włączenia do dokumentacji oddzielone przecinkami.
filename = /home/uzytkownik/Projekt/modul1, /home/uzytkownik/Projekt/modul2

;; Katalogi z kodem źródłowym/przykładami oddzielone przecinkami
directory = /home/uzytkownik/Projekt1/przyklady

;; Katalogi/pliki do ignorowania, oddzielone przecinkami
ignore = /home/uzytkownik/Projekt1/modul1/informacje/*

;; lista layoutów dokumentacji, oddzielona przecinkami.
;; wartosci: HTML:frames:default,HTML:frames:l0133t,
;; HTML:frames:phpdoc.de,HTML:frames:phphtml1lib
;; HTML:frames:DOM/default,HTML:frames:DOM/l0133t,
;; HTML:frames:DOM/phpdoc.de,HTML:Smarty:default
;; HTML:Smarty:PHP,HTML:Smarty:Hands,
;; PDF:default:default,CHM:default:default,
;; XML:DocBook/peardoc2:default
output=HTML:frames:phpdoc.de,PDF:default:default
```

Aby załadować konfigurację, wystarczy wskazać gotowy plik w interfejsie WWW w zakładce „Config”, albo wywołać *phpdoc* z opcją *-c*:

```
phpdoc -c /sciezka/do/konfiguracji.ini
```

Pamiętaj, aby w konfiguracji uwzględnić wszystkie niezbędne opcje, gdyż w przeciwnym razie dodatkowe, ręcznie wpisane ustawienia będą ignorowane.

Dołączanie artykułów

Zwyczajna lista funkcji i klas najczęściej nie wystarczy użytkownikom, szczególnie nowym w temacie, którzy oczekują jakiegoś wprowadzenia i porady dotyczącej zastosowania udostępnianych elementów. Możemy z tym sobie poradzić, dołączając zewnętrzne artykuły napisane w XML-owej aplikacji o nazwie DocBook. Gotowe artykuły możemy dodawać na trzech różnych poziomach:

1. Na poziomie pakietów (pliki *nazwapakietu.pkg*)
2. Na poziomie klas (pliki *nazwaklasy.cls*)
3. Na poziomie plików (pliki *nazwapliku.php.proc*)

Wszystko musi być zorganizowane w odpowiedniej strukturze katalogowej. Zaczniemy od utworzenia nowego katalogu i dopisania go do listy folderów, które ma przeglądać phpDocumentor („Directories to parse”). Następnie stworzymy w nim osobne podkatalogi dla każdego pakietu z osobna. Wszystkie pliki z artykułami zapisujemy w tym pakiecie, którego one dotyczą.

Zajmijmy się teraz samym DocBookiem. Aplikacja ta jest niezwykle rozbudowana i udostępnia kilkaset różnych znaczników do tworzenia zarówno pojedynczych artykułów, jak i całych książek. My raczej nie będziemy w naszej dokumentacji rozwodzić się przez 300 stron tekstu o jednym zagadnieniu, jednak strukturę książki możemy wykorzystać do utworzenia bardziej złożonego artykułu. Poniżej przedstawiam dwa przykładowe kody:

```
<?xml version="1.0" encoding="ISO-8859-2" ?>
<!DOCTYPE article PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<article>
<artheader>
  <title>Mój artykuł</title>
  <!-- tak możemy oznaczyć autora -->
  <author><honorific>Mgr inż.</honorific>
    <firstname>Jan</firstname><surname>Kowalski</surname></author>
</artheader>

<para>Tutaj treść jakiegoś paragrafu</para>

<sect1><title>Tytuł sekcji 1</title>
  <para>Paragraf pierwszy sekcji 1</para>

  <sect2><title>Tytuł sekcji 2</title>
    <para>Paragraf pierwszy sekcji 2</para>
    <!-- i tak możemy zagłębiać sobie do znacznika "sect5" -->
  </sect2>

  <para>Paragraf drugi sekcji 1</para>
</sect1>
<para>Treść jakiegos paragrafu</para>
</article>
```

Do tworzenia paragrafów wykorzystujemy znacznik `<para>`, możemy także bardzo wyraźnie oznaczać poszczególne sekcje rozdziałów, nadając każdej z nich oddzielny tytuł. Przy pisaniu artykułów w DocBooku, należy pamiętać, że każdy dokument musi być w pełni poprawnym plikiem XML.

Pojedynczy rozdział wygląda następująco:

```
<?xml version="1.0" encoding="ISO-8859-2" ?>
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<chapter><title>Tytuł rozdziału</title>
  <para> treść paragrafu </para>
```

```

<sect1><title>Sekcja 1</title>
  <para>Z sekcjami jest tu tak samo, jak w przypadku artykułu</para>
</sect1>
</chapter>

```

Wreszcie, możemy utworzyć kompletną książkę:

```

<?xml version="1.0" encoding="ISO-8859-2" ?>
<!DOCTYPE book PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<book>
<bookinfo>
<title>Moja pierwsza książka</title>
<author>
  <firstname>Jan</firstname>
  <surname>Kowalski</surname>
</author>
<copyright>
<year>2009</year>
<holder>Tomasz Jędrzejewski</holder>
</copyright>
</bookinfo>
<preface>
<title>Tytuł przedmowy</title>
  <para>Treść przedmowy w paru paragrafach</para>
</preface>
<!-- tutaj rozdziały, konstruowane tak samo, jak w poprzednim przykładzie -->
<chapter> ... </chapter>
<chapter> ... </chapter>
<chapter> ... </chapter>

<!-- a tu są dodatki -->
<appendix><title>Dodatek A</title>
  <para>treść dodatku</para>
</appendix>
<appendix><title>Dodatek E</title>
  <para>treść dodatku</para>
</appendix>
</book>

```

Aby do artykułu dołączyć przykładowy kod skryptu, używamy znacznika `<example>`:

```

<example>
  <title>Tytuł przykładu</title>
  <programlisting><![CDATA[<?php
    echo 'Jakiś przykład';
  ]]></programlisting>
</example>

```

Tworzenie list numerowanych i wypunktowanych:

```

<orderedlist>
  <listitem>Element 1</listitem>
  <listitem>Element 2</listitem>
  <listitem>Element 3</listitem>
</orderedlist>
<itemizedlist>
  <listitem>Element 1</listitem>
  <listitem>Element 2</listitem>
  <listitem>Element 3</listitem>
</itemizedlist>

```

W obrębie elementów również możemy stosować paragrafy i inne elementy formatowania tekstu.

Do wyróżniania tekstu mamy znacznik `<emphasis>`, zaś do wymienianych w tekście nazw zmiennych, klas itd. możemy wykorzystać całą gamę semantycznych znaczników w stylu `<filename>` czy `<classname>`. Dokładny opis wszystkich dostępnych znaczników znajdziesz na stronie <http://www.docbook.org>.

Jeżeli do danego elementu chcemy zamieścić więcej poradników, tworzymy plik o nazwie np. *nazwapakietu.pkg.ini* (w przypadku pakietów) z treścią:

```
[Linked tutorials] tutorial1 tutorial2 tutorial3 tutorial4 tutorial5
```

Wszystkie „tutorialxxx” są nazwami plików z artykułami, tyle że bez podanych rozszerzeń. PhpDocumentor automatycznie je odnajdzie i dołączy do dokumentacji.

Porady praktyczne

Poznaliśmy tworzenie dokumentacji z phpDocumentorem od strony technicznej. Wiemy, jak zamieszczać opisy do poszczególnych elementów oraz jak dodawać artykuły, jednak to jeszcze nie wszystko, aby utworzyć dobrą i przyjazną dokumentację. Bycie dobrym programistą wcale nie implikuje umiejętności ciekawego dokumentowania pisanego kodu. Nie liczy się tu błyskotliwość technologiczna czy projektowa, a zwykła zdolność do rzeczowego przelania myśli w tekst tak, by druga osoba potrafiła zrozumieć to, co napisaliśmy, co nauczane jest w szkołach na języku polskim. Podobnie jak nie zostaje się dobrym informatykiem, po prostu czytając jakąś książkę, tak i tutaj żaden artykuł nie podniesie merytorycznego poziomu tekstu. Konieczne są żmudne ćwiczenia i stopniowe poprawianie warsztatu.

Z pewnością można zadać pytanie, co dokumentować, by czytelnik znalazł tam wszystko, czego potrzebuje? Jak kilkakrotnie wspominałem, suchy opis metod i klas nie wystarczy, gdyż bardzo łatwo się w nim zgubić. Konieczne jest dodanie jakiegoś cyklu przewodników stopniowo wprowadzających nowych programistów w środowisko. Nie należy od razu zalewać czytelnika technicznymi szczegółami – bardzo często warto pominąć sporo detali, by nie zaciemniać obrazu całości i wprowadzić je dopiero później, gdy będzie już znana ogólna zasada działania. Na podobnej zasadzie działa nauczanie w szkołach; wprowadzane jest ogólne pojęcie z pewnymi zagadnieniami przyjętymi jako pewnik lub na wiarę, a rozwijane są one dopiero w późniejszym okresie. Nowy programista musi być też zaznajomiony z terminologią oraz powiązaniem między poszczególnymi komponentami. Z pewnością doceni dużą ilość przykładów w postaci krótkich kawałków kodu dostępnych do wykorzystania i omówionych, dlaczego zastosowany został taki, a nie inny wariant.

Nie wszystkie elementy projektu da się opisać w postaci dokumentacji API. Sam phpDocumentor musiał wykorzystać dodatkowe artykuły, aby omówić chociażby listę dostępnych tagów czy interfejs wiersza poleceń. W tym wypadku nie mamy wyjścia i musimy uciekać się do pisania standardowego tekstu. Żadne narzędzie nie napisze go za nas, a co najwyżej ubierze nam go w ładną szatę graficzną i doda nawigację.

Jak wspominaliśmy na początku, phpDocumentor tworzony jest z myślą o dokumentowaniu API i cała struktura nawigacyjna generowanych dokumentacji zaprojektowana została pod tym kątem. Możliwość dołączania artykułów stanowi bardziej dodatek, z którego wiele projektów nie korzysta, gdyż podręczniki użytkownika o wiele wygodniej jest pisać w innych narzędziach. Za wzór można postawić Zend Framework, gdzie dokumentacja API w phpDocumentorze jest szczegółowym wykazem wszystkich klas i funkcji, a każdy wchodzący w jego skład komponent został przedstawiony w postaci oddzielnego pakietu. Odnalezienie listy argumentów potrzebnej funkcji jest dzięki temu dość proste. Obszerny poradnik pokazujący praktyczne użycie każdego komponentu został w całości napisany w DocBooku i jest przetwarzany na HTML za pomocą zupełnie innego zestawu narzędzi. Temu zagadnieniu przyjrzymy się szczegółowo w drugim artykule z serii poświęconym dokumentowaniu projektów.

Warto zastanowić się nad językiem dokumentacji. Jeśli tworzymy projekt na wewnętrzne potrzeby firmy, raczej nie ma przeciwwskazań, aby dokumentacja do niego powstawała w języku polskim, chyba że posiada ona także oddziały zagraniczne. W przeciwnym wypadku nie ma co się wysilać na techniczny angielski, kiedy o wiele lepiej można to samo wyrazić w rodzimej mowie. Nieco inaczej sprawa wygląda z projektami open-source. Jeśli okażą się one przyzwoite, na pewno dowiedzą się o nich obcokrajowcy i aby nie dokładać sobie roboty w przyszłości, najlepiej od razu napisać wszystko po angielsku. Zwróćmy uwagę, że w przypadku umieszczanych w kodzie komentarzy nie mamy możliwości stworzenia kilku wersji językowych bez tworzenia kilku wersji kodu źródłowego, co raczej nie wchodzi w grę. Dlatego warto zadbać o to już na pierwszym etapie prac, by później nie utknąć w procesie tłumaczenia.

Na koniec pewna uwaga techniczna. PHP jest językiem interpretowanym, dlatego nie powinniśmy przesadzać z objętością komentarzy. Choć ich treść nie jest w ogóle analizowana, ilość i objętość już ma pewne odbicie w wydajności. Osobiście w przypadku *phpdoc* ograniczam się do podania w komentarzach jedynie najważniejszych rzeczy, opisanych lakonicznie, zaś bardziej szczegółowe omówienie z przykładami

opracowuję w innym narzędziu, gdzie przy okazji mogę przetłumaczyć je na inne języki. Niemniej z *phpdoc* nie powinno się rezygnować, właśnie dzięki możliwości szybkiego wygenerowania aktualnej dokumentacji doskonale odzwierciedlającej budowę kodu, a także możliwościom podpowiadania oferowanym przez zaawansowane środowiska programistyczne, co z pewnością zostanie docenione przez wykorzystujących je programistów.

Zakończenie

Dokumentowanie kodu to niezwykle obszerny temat. W artykule tym poznaliśmy jedno z kilku narzędzi, które mogą nam w tym pomóc, a także nieco praktycznych wskazówek o ich pisaniu. Mam nadzieję, że temat okazał się interesujący i będziesz dalej poszerzać swoją wiedzę na ten temat. Zachęcam również do przeczytania kolejnego artykułu pt. „Dokumentowanie projektu PHP w TypeFriendly”.

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na **www.zyxist.com**

Licencja

Artykuł rozpowszechniany jest na licencji **Creative Commons Uznanie autorstwa-Użycie niekomercyjne-Bez utworów zależnych 2.5 Polska**.

Wolno:

- kopiować, rozpowszechniać, odtwarzać i wykonywać utwór

Na następujących warunkach:

1. *Uznanie autorstwa*. Utwór należy oznaczyć w sposób określony przez Twórcę lub Licencjodawcę*.
2. *Użycie niekomercyjne*. Nie wolno używać tego utworu do celów komercyjnych.
3. *Bez utworów zależnych*. Nie wolno zmieniać, przekształcać ani tworzyć nowych dzieł na podstawie tego utworu.

W celu ponownego użycia utworu lub rozpowszechniania utworu należy wyjaśnić innym warunki licencji, na której udostępnia się utwór.

Każdy z tych warunków może zostać uchylony, jeśli uzyska się zezwolenie właściciela praw autorskich.

Powyższe postanowienia w żaden sposób nie naruszają uprawnień wynikających z dozwolonego użytku ani żadnych innych praw.

Internetowa skrócona wersja licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/>

Pełen tekst licencji:

<http://creativecommons.org/licenses/by-nc-nd/2.5/pl/legalcode>

Podczas publikacji należy podać następujące informacje:

1. Link do internetowej skróconej wersji licencji.
2. Informację o wersji publikowanego tekstu.
3. Informacje o autorze w następujący sposób:

Tomasz "Zyx" Jędrzejewski

Aktualna wersja artykułu zawsze na **www.zyxist.com**