

Tomasz "Zyx" Jędrzejewski

Dokumentacje w TypeFriendly

Wersja 1.0 (25 lipca 2008)



Szczegółowe informacje o licencji znajdują się pod artykułem.

www.zyxist.com

Dokumentacje w TypeFriendly

Każdy liczący się projekt powinien posiadać dokumentację użytkownika. Sposobów na jej wykonanie jest wiele – począwszy od ręcznego wstukiwania wszystkiego, aż do rozmaitych skryptów i narzędzi stworzonych do tego celu. Zaletą takich narzędzi jest automatyzacja części zadań, jak np. generowanie nawigacji, kolorowanie kodów źródłowych itd. Do niedawna korzystałem z XML-owej aplikacji nazwanej DocBook. Dokumentacja powstawała tam w formacie XML, a na inne formaty, np. HTML konwertowana była arkuszami transformacji XSLT. Z tego samego rozwiązania korzysta np. firma Zend Company do dokumentowania swoich projektów, w tym m.in. języka skryptowego PHP. Ja jednak byłem tym narzędziem rozczarowany. Oryginalny DocBook i arkusze transformacji same w sobie są lekko wybrakowane – wersja źródłowa musi być pisana w jednym, wielkim pliku, nie ma mowy o zaawansowanej wielojęzyczności (uzupełnianie brakujących rozdziałów tłumaczenia oryginałem) czy choćby kolorowania składni przykładów. Twórcy PHP, aby poradzić sobie z tym wszystkim, napisali specjalny, dodatkowy framework, który uchodzi za trudny w instalacji i jest na dokładkę zależny od systemu operacyjnego.

W tej sytuacji zdecydowałem się porzucić DocBooka i rozglądać się za czymś innym, co nie posiadałoby wad poprzednika. Rozczarowany propozycjami znalezionymi w sieci stwierdziłem, że nie pozostaje mi nic innego, jak opracować własny system wspomagający. Określiłem najważniejsze zadania. Skrypt musiał umożliwiać:

1. Pisanie każdego rozdziału w osobnym pliku.
2. Ręczne ustawianie kolejności rozdziałów tam, gdzie to jest potrzebne, a w pozostałych przypadkach samodzielnie ustawiać je w porządku alfabetycznym.
3. Generowanie spisów treści i nawigacji między rozdziałami.
4. Zaawansowane opcje formatowania tekstu, w tym kolorowanie składni kodu źródłowego.
5. Wsparcie dla dokumentacji wielojęzycznych: narzędzie do sprawdzania, które pliki oryginału uległy zmianie i nie zostały poprawione w tłumaczeniu oraz uzupełnianie brakujących rozdziałów tłumaczenia tekstem oryginalnym.
6. Niezależność od systemu operacyjnego.

Na parser składni wybrałem system Markdown cechujący się wyjątkową prostotą i elegancją – w praktyce tekst źródłowy jest równie czytelny, jak wyjściowy. Ponadto, przyjąłem, że systemowi wystarczy sama nazwa pliku, aby określić położenie rozdziału i jego relację w stosunku do innych części tekstu. Tak narodził się TypeFriendly, a jego pierwsza wersja użytkowa ukazała się pod koniec lipca 2008.

Instalacja

TypeFriendly napisany jest w PHP, dlatego do działania wymaga zainstalowanego interpretera tego języka. Czysty PHP jest niezwykle prosty w instalacji. Na systemach Linuksowych jest to kwestia skorzystania z menedżera pakietów, zaś pod Windowsem wystarczy odwiedzić www.php.net, ściągnąć najnowszą wersję PHP 5.2.x, rozpakować na dysku i w Panelu Sterowania ustawić zmienną środowiskową `PATH` tak, by znalazła się w niej ścieżka do pliku `php.exe`.

TypeFriendly możemy pobrać ze strony www.invenzzia.org (dział „Pliki”). Instalacja sprowadza się do rozpakowania skryptu gdzieś na dysk twardy. Narzędzie obsługiwane jest w całości z wiersza poleceń systemu operacyjnego, zaś najprostszym sposobem, aby przekonać się, czy działa, jest próba wygenerowania jego własnej dokumentacji, dostarczonej właśnie w wersji źródłowej i służącej jednocześnie za przykład:

```
php typefriendly.php -o xhtml ./docs/
```

Jeśli wszystko poszło dobrze, w folderze `/docs/output/xhtml/` powinieneś znaleźć polskojęzyczną dokumentację do TypeFriendly w formacie XHTML.

Zaczynamy

Rozpocznijmy teraz prace nad własną dokumentacją. Weź jakiś swój projekt i utwórz w nim katalog na dokumentację zwany dalej folderem projektu. TypeFriendly wymaga odpowiedniej struktury na jej przechowywanie. Zacznij od stworzenia bezpośrednio w folderze projektu pliku *settings.ini*, który zawierać będzie podstawowe informacje i ustawienia. Jego składnia jest niezwykle prosta i składa się z par **nazwa="wartość"** umieszczonych w kolejnych liniijkach. Dodatkowo, znak średnika rozpoczyna komentarz. Oto przykładowy plik:

```
title = "Mój projekt"
version = "1.0"
copyright = "Moje dane"
copyrightLink = "http://www.example.org/"
license = "GNU Free Documentation License 2.1"
licenseLink = "http://www.gnu.org/licenses/fdl.html"

; Ustawienia funkcjonalne projektu

outputs = "xhtml, xhtml_single"
baseLanguage = "pl"
navigation = "book"
showNumbers = true
```

Pierwsze sześć ustawień to kilka podstawowych wiadomości o Twoim projekcie i dokumentacji. Podaj jego nazwę, wersję oraz informacje o prawach autorskich. W TypeFriendly niezbędne jest również określenie licencji – jeśli robisz projekt open-source, warto pozostawić FDL, w przeciwnym razie wybierz to, co Ci najbardziej odpowiada.

Druga część to rozmaite ustawienia funkcjonalne. W TypeFriendly za utworzenie ze źródeł wynikowych plików odpowiadają tzw. systemy wyjścia. Te, których chcemy używać w naszym projekcie, musimy wymienić w dyrektywie *outputs*, oddzielając ich nazwy przecinkiem. Obecnie lista dostępnych nie jest jeszcze długa i obejmuje jedynie XHTML oraz XHTML na jednej stronie. W przygotowaniu jest generator wersji do umieszczenia w bazie danych, co może być przydatne przy tworzeniu wersji on-line. Niestety, część ograniczeń narzuca tu używany aktualnie parser Markdowna, który niczego poza XHTML-em nie wspiera i dopóki nie zostanie on przepisany, sytuacja nie ulegnie poprawie.

Musisz także wybrać podstawowy język, w którym tworzona będzie dokumentacja. Jest to część systemu obsługi tłumaczeń, który omówimy dalej. Ostatnie dwie opcje dotyczą sposobu organizowania nawigacji oraz pokazywania numerów rozdziałów. Nawigacja w stylu książkowym powoduje, że odnośniki „Poprzedni” oraz „Następny” pozwalają na podróż od początku do nawigacji, tak jak w książce i jak domyślnie robi DocBook. Alternatywna wartość, *tree*, zachowuje drzewiasty charakter rozdziałów. Najlepiej wypróbuj oba warianty i spróbuj zobaczyć różnicę na własne oczy.

Piszemy rozdział

Pora napisać pierwszy rozdział. W tym celu musisz założyć w folderze projektu katalog */input/pl/*, w którym umieścisz wersję źródłową z polskim tekstem. Dla każdego rozdziału tworzymy osobny plik o rozszerzeniu TXT. TypeFriendly wykorzystuje nazwy plików nie tylko do identyfikowania rozdziałów, ale też do ustalenia zależności między nimi (tzn. czy A jest podrozdziałem B), dlatego istnieją ścisłe zasady nazewnictwa. Będziemy je poznawać stopniowo.

Dokumentacja powinna zawierać wstęp z podstawowymi informacjami propagandowo-marketingowymi o naszym projekcie. Umieścimy go w pliku *preface.txt*:

```
Title: Wstęp

-----

Witam w dokumentacji Mojego Projektu. Został on stworzony na potrzeby
artykułu o systemie dokumentacji [TypeFriendly], aby było co dokumentować.
```

Cechy tego projektu to:

1. Nie istnieje.
2. Nie można go znikąd ściągnąć.
3. Nic nie robi.

Mam nadzieję, że się podoba.

```
[TypeFriendly]:  
    http://www.invenzzia.org/ "TypeFriendly"
```

Przedmowa jest jednym z głównych rozdziałów, dlatego nazwa pliku składa się po prostu z angielskiego, zwężłego identyfikatora i rozszerzenia. Sama treść jest podzielona na dwie części:

1. Nagłówek, w którym umieszczamy tzw. **tagi**. Definiują one różne ustawienia danego rozdziału. W tym wypadku chcemy określić jedynie polski tytuł, a służy do tego właśnie **Title**. Składnia jest bardzo prosta i nietrudno się w niej połapać.
2. Treść, oddzielona od nagłówka paroma kreskami. Sporządzamy ją w formacie Markdown, który jest wyczerpująco omówiony w dokumentacji TF. Tutaj pragnę zwrócić uwagę na jego wyjątkową czytelność. Paragrafy tworzy się bardzo prosto, wystarczy oddzielić je od siebie pustą linią. Bardzo łatwo można tworzyć listy numerowane, z tym że Markdown ignoruje i tak poprawność naszej numeracji (równie dobrze moglibyśmy wszędzie wpisać jedynek lub losowe liczby). To od nas zależy, czy chcemy mieć schludne źródła, czy nie. Ostatnia rzecz warta wzmianki to możliwość tworzenia odnośników przez referencje, dzięki czemu właściwy adres URL nie płącze się w tekście. W pierwszym paragrafie umieściliśmy słowo *TypeFriendly* w nawiasach kwadratowych, zaznaczając, że ma stać się ono odnośnikiem. Parser domyślnie wykorzystał je jako identyfikator, gdyż nie określiliśmy jawnie własnego. Dopiero na końcu dokumentu, w dwóch ostatnich liniach napisaliśmy, do jakiego adresu ma prowadzić odnośnik i jaki tytuł mu nadać.

Spróbujmy teraz stworzyć tekst opisujący instalację naszego projektu zbudowany z dwóch podrozdziałów. W pierwszym opiszemy tzw. instalację prostą dla opornych, zaś w drugiej znajdą się informacje o zaawansowanych opcjach instalacyjnych. Pierwszy plik nazwiemy *installation.simple.txt*, a drugi *installation.advanced.txt*. Tym razem każda nazwa składa się już z dwóch części oddzielonych kropką. Ostatnia zawsze identyfikuje konkretny rozdział, zaś wszystkie poprzednie (w naszym wypadku – *installation*) definiują rozdziały nadrzędne, z którymi chcemy związać aktualny plik. Jednak w takim wypadku TypeFriendly wymaga, aby istniał również plik *installation.txt* i to od niego zaczniemy nasze prace:

```
Title: Instalacja  
  
-----  
  
W tym rozdziale opisane zostaną kwestie instalacji.  
  
Wymagania sprzętowe  
=====
```

1. Procesor Core 2 Quad 7 GHz
2. Karta graficzna z PixelShader 6.0 i 2 GB pamięci
3. Wiatraczek

Tutaj zbyt dużo nie ma, gdyż właściwe treści napiszemy dopiero w podrozdziałach. TypeFriendly odkryje, że z tym plikiem związane są dwa inne i automatycznie doda do niego spis treści podrozdziałów. Przejdźmy jeszcze do Markdowna, gdyż pojawił się tutaj kolejny element składni – nagłówki. Pokazany powyżej przykład jest jedną z dwóch dostępnych w MD implementacji tego elementu i umożliwia stworzenie nagłówków poziomu 2 i 3 (TF rezerwuje poziom 1 na własny użytek). Tekst, który ma stać się nagłówkiem, podkreślamy wizualnie w źródle znakami równości, a w przypadku poziomu 3, pauzami.

Treść pliku *installation.simple.txt* przedstawia się następująco:

```
Title: Instalacja prosta
SeeAlso:
- installation.advanced
```

Ten rozdział opisuje prostą instalację.

1. Rozpakuj pliki ze ściągniętego archiwum.
2. Skopiuj je do jakiegoś folderu, np. `~/usr/local/projekt/``.
3. Wywołaj z linii komend skrypt instalacyjny:

```
[console]
# ./install.sh
```

4. Projekt zainstalowany.

W nagłówku pojawił nam się kolejny tag, **SeeAlso**. Służy on do stworzenia sekcji „Zobacz także” na końcu rozdziału. Jako argument przyjmuje listę identyfikatorów rozdziałów, do których trzeba utworzyć odnośniki. Każdy z nich piszemy w nowej linijce i rozpoczynamy pauzą, dokładnie tak, jak powyżej. Oprócz tego, TF posiada tag **SeeAlsoExternal**, w którym można określić odnośniki zewnętrzne.

Z kolei w treści pojawiła się kolejna lista numerowana, lecz tym razem poszczególne elementy oddzielone są pustymi linijkami. To znak dla Markdowna, że każdy element może zawierać kilka elementów blokowych, np. paragrafy, inne listy czy bloki kodu źródłowego. W podpunkcie drugim ścieżkę napisaliśmy w odwrotnych apostrofach. To standardowy element składni aplikujący czcionkę o stałej szerokości znaków na lekko wyróżniającym się tle. Nadaje się idealnie do umieszczania w tekście ścieżek oraz bardzo krótkich partii kodu źródłowego (np. nazw funkcji). Natomiast w podpunkcie trzecim umieściliśmy fragment kodu do wpisania w konsoli systemowej. Jest on odsunięty dość daleko od lewego marginesu:

1. Pierwsze cztery spacje biorą się z tego, że umieszczamy go w obrębie listy.
2. Kolejne cztery - ze składni samego bloku kodu.

W pierwszej linijce w nawiasach kwadratowych możemy określić rodzaj języka (np. „php”, „cpp”), co nada kodowi odpowiednie kolorowanie składni. Markdown oferuje jeszcze jedną składnię bloków kodu, nadającą się idealnie do dłuższych partii i niewymagającą poprzedzania każdej linijki czterema spacjami. Poznamy ją niebawem.

A oto i drugi z rozdziałów instalacyjnych, *installation.advanced.txt*:

```
Title: Instalacja zaawansowana
SeeAlso:
- installation.simple
```

Ten rozdział opisuje zaawansowaną instalację.

W zasadzie wygląda ona podobnie, jak instalacja prosta, lecz zamiast pliku `install.sh`` należy uruchomić `install_adv.sh``. Skrypt ten zawiera rozbudowany instalator, który pozwoli na manualne ustawienie wszystkich aspektów projektu.

Nie pojawiło się tutaj nic odkrywczego.

Opisujemy API

Kolejne rozdziały będą zawierać opis API naszego projektu. Będzie tu okazja do zaznajomienia się z kilkoma

narzędziami pomocniczymi, w jakie został wyposażony TypeFriendly. Projekt nie potrafi jeszcze generować odpowiednich plików na podstawie struktury kodu źródłowego tak, jak to robi np. phpDocumentor, ale opcja taka jest już w planach. Póki co musimy tworzyć wszystko ręcznie, lecz na szczęście nie jest to takie skomplikowane. Utworzenie załączków rozdziałów dla pięciu klas zawierających po 10 do 15 metod to zajęcie na jedną godzinę, które w dodatku można zautomatyzować jakimś skryptem.

Na początek założymy rozdział główny *api.txt*, do którego trafią opisy wszystkich klas:

```
Title: Opis API
----

Rozdział ten zawiera opis API projektu.

> [warning]
> Do prawidłowego działania wymagane są najnowsze wersje bibliotek.
```

Na końcu treści widzimy blok cytatu (tak on się nazywa w Markdownie) i tak też jest domyślnie interpretowany przez TF. Jeśli jednak w pierwszym wierszu umieścimy nawias kwadratowy, uzyskamy ramkę informacyjną podanego typu. W powyższym przypadku będzie to ramka ostrzegawcza („warning”). TypeFriendly podkreśli tekst delikatnym, czerwonym tłem, a obok umieści ikonkę wykrzyknika.

Założmy teraz rozdziały dla naszych dwóch klas:

api.klasa1.txt:

```
Title: Klasa "Klasa1"
ShortTitle: Klasa1
EExtends: foo
ExtendedBy:
  - api.klasa2
----

Niniejsza klasa realizuje najważniejsze funkcje projektu. Lista pól:
```

Pole	Typ	Opis
pole1	int	Opis pola 1
pole2	int	Opis pola 2
pole3	double	Opis pola 3

Omówmy najpierw szereg nowych tagów, które pojawiły się w nagłówku. **Title** to podstawowy tytuł rozdziału – o tym już wiemy. Wymieniamy w nim rodzaj opisywanego elementu („Klasa”) oraz jej nazwę. Jednak tworząc odnośnik do tego rozdziału, byłoby fajnie, gdyby pokazywała się tam jedynie nazwa klasy. Takie jest właśnie zastosowanie **ShortTitle**, czyli skrócony na potrzeby odnośników tytuł.

Kolejne dwa tagi pomagają w określeniu zależności klas w programowaniu obiektowym. Aktualnie istnieją trzy ich rodzaje, z czego każdy występuje w dwóch wersjach:

1. Przyjmującej identyfikatory rozdziałów dokumentacji TF.
2. Przyjmującej bezpośrednio nazwy elementów zewnętrznych, których nasza dokumentacja nie definiuje. Nazwy tych tagów rozpoczynają się od dużej litery „E”.

W powyższym przykładzie mamy tag **EExtends**, który jest odmianą **Extends**. Pozwala określić rodzica aktualnej klasy, a dodatkowa litera **E** mówi nam, że jest to jakaś klasa zewnętrzna i podajemy jej nazwę bezpośrednio. Z kolei **ExtendedBy** umożliwia podanie, jakie klasy rozszerzają tę, którą właśnie opisujemy. Tym razem na początku nie ma dodatkowej litery **E**, więc w nowych liniach po pauzie wymieniamy nazwy rozdziałów. Nic nie stoi na przeszkodzie, aby użyć jednocześnie **ExtendedBy** oraz **EExtendedBy** w tym samym dokumencie.

W treści pojawił się kolejny element składni Markdowna, którego postać źródłowa ceni sobie czytelny

wygląd. Mianowicie tworzymy tabelkę z opisami pól klasy, dosłownie rysując ją w ASCII przy pomocy pauz i pionowych kresek. Ma to pewne ograniczenia, ale w razie czego parser obsługuje także wstawki HTML-a.

Plik *api.klasa2.txt* prezentuje się następująco:

```
Title: Klasa "Klasa2"
ShortTitle: Klasa2
Extends: api.klasa1

----

Niniejsza klasa realizuje dodatkowe funkcje projektu. Nie dodaje ona żadnych pól.
```

Czas opisać metody tych klas. Zaczniemy od *api.klasa1.foo.txt* czyli metody *foo()* w klasie *klasa1*:

```
Title: foo()
ShortTitle: Klasa1::foo()
Reference: int Klasa1::foo(int arg)

----

Metoda ta realizuje pierwszą funkcję projektu. Jako argument `arg`
przekazujemy liczbę całkowitą. Przykład użycia:

~~~~
[cpp]
obiekt = new Klasa1;
if(obiekt->foo(5))
{
    cout << "OK";
}
~~~~
```

W opisach funkcji i metod bardzo pomocny jest tag **Reference**, w którym możemy zdefiniować składnię wywołania, wymieniając argumenty, ich typy oraz zwracaną wartość. Oczywiście całość nie obejdzie się bez przykładu użycia, który korzysta tym razem z drugiego sposobu osadzania bloków kodu w tekście. Całość ograniczamy tutaj z dwóch stron czterema tyldami, a w pierwszej linijce w nawiasie kwadratowym definiujemy użyty język tak, by TypeFriendly mógł pokolować nam składnię.

Opis drugiej metody w pliku *api.klasa1.bar.txt* jest bardzo podobny:

```
Title: bar()
ShortTitle: Klasa1::bar()
Reference: void Klasa1::bar(int arg1, int arg2)

----

Metoda ta realizuje drugą funkcję projektu. Jako argumenty należy podać dwie
liczby całkowite. Przykład użycia:

~~~~
[cpp]
obiekt = new Klasa1;
if(obiekt->bar(5, 8))
{
    cout << "OK";
}
~~~~
```

Aby i drugiej klasie nie było smutno, dodajmy jakąś metodę również tam: *api.klasa2.joe.txt*

```
Title: joe()
ShortTitle: Klasa2::joe()
Reference: void Klasa2::joe()

-----

Pomocnicza metoda projektu.
```

To już wszystkie pliki naszej dokumentacji. Pozostaje tylko przełączyć się do katalogu z TypeFriendly i z konsoli odpalić generator:

```
./typefriendly.php -o xhtml /sciezka/do/katalogu/z/dokumentacja/
```

a w przypadku użytkowników Windowsa:

```
php typefriendly.php -o xhtml /sciezka/do/katalogu/z/dokumentacja/
```

Ważne jest, aby pamiętać o ukośniku na końcu ścieżki do katalogu projektu. Jeśli wszystko poszło dobrze, w katalogu */output/xhtmll* znajdziesz wygenerowaną dokumentację na wielu stronach HTML.

Zarządzanie kolejnością

Prawdopodobnie już zauważyłeś, że coś tu nie gra. TypeFriendly zawartość wszystkich rozdziałów posortował alfabetycznie, przez co np. wstęp znalazł się na końcu. Oczywiście tak być nie może, ale nie jest to żaden błąd. Po prostu nie określiliśmy żadnych reguł kolejności, więc TF zmuszony był ułożyć wszystko według alfabety. Aby wpłynąć na kolejność, musimy założyć w głównym katalogu projektu plik *sort_hints.txt*:

```
preface
installation
installation.simple
installation.advanced
api
api.klasa1
api.klasa2
```

Jego budowa jest prosta, jak konstrukcja cepa. Najprościej w świecie wymieniamy rozdziały w takiej kolejności, w jakiej chcemy, żeby były. Musimy tylko pamiętać o trzech rzeczach:

1. Rozdziały związane z tym samym rozdziałem nadrzędnym muszą się znaleźć obok siebie (*installation.simple* i *installation.advanced* muszą być obok siebie oraz pod *installation*).
2. Nie musimy wymieniać podrozdziałów jakiegoś rozdziału. Wtedy TypeFriendly pozostawi dla nich porządek alfabetyczny. Tak też zrobiliśmy z opisami metod, gdzie w zupełności on nam wystarczy.
3. Jeśli mamy rozdział A i na liście wymieniliśmy choć jeden jego podrozdział, musimy wymienić też wszystkie pozostałe. Inaczej TypeFriendly zgłosi błąd podczas generowania.

Wskazówki dot. dokumentacji wielojęzycznych

TypeFriendly posiada również wsparcie dla dokumentacji wielojęzycznych. Zawsze musimy wybrać tzw. język bazowy, w którym powstaje oryginalna treść. Wszystkie pozostałe języki traktowane są jak tłumaczenia. Zaczynamy od zrobienia w */input* nowego folderu na kolejny język (najlepiej nazwać go dwuliterowym kodem, gdyż tak TF identyfikuje języki). Potem wystarczy stworzyć tam przetłumaczone rozdziały i gotowe. Jednak pamiętaj, że identyfikatory rozdziałów w oryginale i w tłumaczeniach muszą być takie same! TypeFriendly stosuje bowiem jednolite reguły definiowania porządku do wszystkich języków i tłumaczeń, a ponadto musi mieć możliwość porównania zawartości. Podczas samego generowania realizowany jest

następujący algorytm:

1. Bez względu na to, czy generujemy oryginał czy tłumaczenie, TF buduje strukturę dokumentacji zawsze na podstawie tego, co znajdzie w katalogu języka bazowego.
2. Jeśli istnieje odpowiednik danego rozdziału w tłumaczeniu, wtedy używa właśnie jego.
3. W przeciwnym wypadku bierze treść z oryginału.

Przed założeniem nowego języka upewnij się, że TF zawiera też pliki językowe interfejsu dla niego w katalogu */languages*. Jeśli nie, będziesz musiał rozpocząć od przetłumaczenia właśnie ich. Nie zapomnij opublikować ich w Internecie tak, by i inni mogli z nich skorzystać.

Zakończenie

TypeFriendly stworzyłem z myślą o wygodzie i elegancji, której brakowało innym rozwiązaniom. Wreszcie prace organizacyjne przy zakładaniu nowej dokumentacji stały się krótką formalnością. Z rezultatu jestem zadowolony, choć z pewnością trochę opcji projektowi jeszcze się przyda. Zachęcam do zapoznania się z tym narzędziem oraz, w miarę możliwości, pomocy w jego rozwoju.

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