

*Tomasz "Zyx" Jędrzejewski*

# Wprowadzenie do Open Power Template

część 1

Wersja 1.0 (7.10.2006)



Szczegółowe informacje o licencji znajdują się pod artykułem.

**[www.zyxist.com](http://www.zyxist.com)**

# Wprowadzenie do OPT cz. 1

Idea separacji mechanizmów prezentacji i przetwarzania danych powraca wśród programistów PHP, jak bumerang. Nikt nie wątpi, że powinny być one oddzielone od siebie: programista pracuje nad swoją częścią, webmaster tworzy kod HTML i nikt nikomu nie wchodzi w drogę, gdyż wszystko jest w osobnych plikach. Dyskusje powstają w momencie, kiedy zaczyna się mówić o formie i wykonaniu. Jedną z koncepcji, przez jednych wychwalaną, przez innych krytykowaną, są systemy szablonów. Są to specjalne biblioteki, które potrafią przetwarzać pliki HTML ze specjalnymi znacznikami mówiącymi, gdzie mają się pokazywać dane generowane przez skrypt.

Niekwestionowanym królem systemów szablonów, jeśli chodzi o popularność, jest Smarty™. Rozwijany jest on od bardzo dawna, co z punktu widzenia stabilności jest korzystne, lecz ma przełożenie na możliwości. Wielu programistów krytykuje go za zbytne zbliżanie się do języka programowania, "niezyciowe" nazewnictwo metod w formie "nazwa\_nazwa()", zamiast bardziej "eleganckiego" camel style, czy nawet ograniczenia samej architektury. Smarty przez długi czas miał też problemy z działaniem na PHP 5.

Artykuł ten koncentruje się jednak na innym systemie szablonów, stworzonej w Polsce bibliotece Open Power Template. Na pierwszy rzut oka może się ona wydawać klonem Smarty'ego, lecz bliższe spotkanie ujawnia mnóstwo różnic, zarówno w składni, możliwościach, jak i architekturze. Oto krótka charakterystyka biblioteki:

1. OPT można używać wyłącznie na serwerach wyposażonych w PHP 5 (stale śledzony jest także rozwój PHP6 tak, by już na tym etapie zapewnić kompatybilność w przód).
2. OPT oferuje nie tylko niskopoziomowe instrukcje, mające swe odpowiedniki w językach programowania. Posiada też szereg bardziej intuicyjnych instrukcji.
3. OPT posiada wbudowane wsparcie dla mechanizmów i18n.
4. OPT posiada mechanizm komponentów, przydatny przy budowaniu dynamicznych formularzy.
5. Wiele dodatkowych opcji, np. automatyczne zarządzanie nagłówkami HTTP, kompresja gZip, system cache, tryb emulacji XML.
6. OPT w testach wydajnościowych wygrywa ze Smarty'm, chociaż algorytmy ich działania oparte są na identycznej idei prekompilowania szablonów do postaci kodu PHP. Wydajność można poprawić jeszcze bardziej, korzystając z akceleratorów, np. *eAccelerator*, *APC* czy *Zend Optimizer*.
7. OPT posiada otwartą architekturę - możliwe jest nawet napisanie własnego parsera opartego o kompilator OPT, dzięki OPT API.

Przy projektowaniu biblioteki cały czas brane było pod uwagę, co w Smarty'm można by zrobić inaczej, lepiej, bardziej intuicyjnie. W pracach pomagała też polska społeczność PHP, testując wczesne wersje i podsuwając wiele cennych pomysłów.

Open Power Template jest częścią większego projektu obejmującego zarówno aplikację forum dyskusyjnego Open Power Board, jak i zestaw pokrewnych bibliotek (Open Power Driver - nakładka na PHP Data Objects, Open Power Forms - dodatek do OPT koncentrujący się na zarządzaniu formularzami).

## Instalacja

Instalacja OPT jest bardzo prosta. Na początku wchodzimy na witrynę WWW projektu: <http://opt.openpb.net/> i pobieramy stamtąd najnowszą wersję (kolejne wydania ukazują się na początku każdego miesiąca). W ściągniętym archiwum znajduje się katalog "lib", którego zawartość musimy skopiować w drzewo katalogowe naszego własnego projektu. Jeśli zamierzamy korzystać z pluginów, możemy utworzyć gdzieś katalog "plugins".

Kolejnym krokiem jest utworzenie dwóch katalogów na szablony: *templates* będzie zawierał wersje źródłowe szablonów, czyli takie, które utworzysz ty sam, natomiast *templates\_c* posłuży do przechowywania skompilowanych wersji, generowanych przez OPT. Oczywiście nazwy katalogów i ich dokładna lokalizacja są całkowicie dowolne. W systemach uniksowych musimy pamiętać o nadaniu skryptowi praw do zapisu dla katalogu *templates\_c* - inaczej powstaną błędy przy próbie zapisu skompilowanych szablonów. To wszystko.

Pozostało jedynie zainicjowanie skryptu.

## Pierwszy skrypt

Open Power Template został napisany z wykorzystaniem technik programowania obiektowego, dlatego od jego użytkownika wymagana jest pewna znajomość takich pojęć, jak klasa, metoda, czy obiekt i właśnie nimi będziemy operować w dalszej części tekstu.

Zacniemy od stworzenia szablonu HTML, który umieścimy w katalogu *templates*. Przyjęło się nadawanie plikom szablonów rozszerzenia *.tpl*.

```
<html>
<head>
  <title>Open Power Template: pierwszy skrypt</title>
</head>
<body>
  <p>Hello world! Dzisiaj jest {$obecna_data}</p>
</body>
</html>
```

Tak, jak wspominaliśmy, system szablonów pobiera z szablonu kod HTML i odnajduje w nim specjalne znaczniki, które następnie zamieniane są na dane ze skryptu. W powyższym przykładzie takim znacznikiem jest napisana pogrubioną czcionką *{\$obecna\_data}*. Klamry ograniczają zasięg znacznika. *\$obecna\_data* to tzw. blok, reprezentujący miejsce, w którym umieszczamy dane ze skryptu. Napiszmy teraz skrypt, który będzie nakazywać przetworzenie naszego szablonu:

```
<?php
define('OPT_DIR', './lib/');
require('./lib/opt.class.php'); // 1

try
{
    $tpl = new optClass; // 2
    $tpl -> root = './templates/';
    $tpl -> compile = './templates_c/';
    $tpl -> gzipCompression = true;
    $tpl -> httpHeaders(OPT_HTML); // 3

    $tpl -> assign('obecna_data', date('d.m.Y, H:i')); // 4
    $tpl -> parse('szablon1.tpl'); // 5
}
catch(optException $exception)
{
    optErrorHandler($exception); // 6
}
?>
```

Przyjrzyjmy się szczegółowo skryptowi:

1. Na początku musimy dołączyć do skryptu plik *opt.class.php*, przedtem definiując stałą *OPT\_DIR* zawierającą ścieżkę do plików biblioteki. Zaleca się podawanie jej właśnie w takiej postaci, jak na przykładzie, tj. zaczętej od *./* - dzięki temu PHP nie będzie przeszukiwać kilkunastu domyślnych ścieżek w dyrektywie *include\_path*, co negatywnie odbija się na wydajności.
2. Tutaj tworzymy obiekt klasy *optClass*, czyli samego parsera szablonów.
3. Ta bardzo ciekawa metoda automatycznie wysyła nagłówki HTTP dla ustalonego typu zawartości. Jeśli dodatkowo w konfiguracji ustawimy dyrektywę *charset*, wyśle automatycznie informacje o kodowaniu (jest to szczególnie ważne, gdy chcemy stworzyć witrynę wykorzystującą UTF-8).
4. Pomostem między skryptem, a szablonem, jest metoda *assign()*. To za jej pomocą powiadamy OPT, jakie dane mamy zamiar umieścić w szablonie oraz pod jakimi nazwami będą one dostępne. W tym wypadku zapisujemy pod nazwą *obecna\_data* aktualną datę.
5. Ostatnim krokiem jest wywołanie naszego szablonu. OPT wczyta go i połączy z danymi, a gotowy

kod HTML automatycznie wyśle do przeglądarki internauty.

6. Ewentualne błędy są zgłaszane jako wyjątki. W tym miejscu musimy je przechwycić. Jeśli nie mamy własnego handlera, możemy skorzystać z gotowego `optErrorHandler()`, który automatycznie sformatuje komunikat.

Gratulacje, właśnie napisałeś swój pierwszy skrypt wykorzystujący OPT!

## Listy

Open Power Template to nie tylko prymitywne umieszczanie danych w szablonie. Biblioteka udostępnia szereg narzędzi pozwalających na zaawansowaną manipulację informacjami. W praktyce po stronie szablonu mamy do swej dyspozycji całkiem niezły język programowania, z obsługą zmiennych, wyrażeń, pętli oraz instrukcji warunkowych. Jednak należy to traktować bardziej jako ciekawostkę, ponieważ wraz z nim dostajemy również instrukcje pozwalające choćby częściowo o programowaniu zapomnieć, które zautomatyzują wiele czynności. Jedną z nich są *sekcje*. Służą one do generowania wszelkiego rodzaju list za pomocą niezwykle prostej składni. Poniższy przykład pokazuje, jak stworzyć listę albumów muzycznych.

Zaczynamy od szablonu. Stworzenie sekcji polega po prostu na określeniu wyglądu pojedynczego elementu listy - resztą zajmuje się OPT:

```
<html>
<head>
  <title>Open Power Template: albumy muzyczne</title>
</head>
<body>
  <h3>Moja lista albumów</h3>

  <ul>
    {section=album}
    <li>{$album.tytul} - {$album.zespol} ({$album.rok})</li>
    {/section}
  </ul>
</body>
</html>
```

Nasz element ograniczony jest znacznikami `{section=album}` (tu określamy też nazwę sekcji - musi być ona unikalna, abyśmy mogli podpiąć do niej później dane listy) oraz zamykającym `{/section}`. Aby wstawić w jakimś miejscu dane elementu, używamy specjalnego bloku sekcji, np. `{$album.tytul}`. Jak widać, zbudowany jest on z dwóch części oddzielonych kropką. Pierwsza to nazwa naszej sekcji, druga to nazwa jednego z bloków niesionych przez dany element.

Przejdźmy teraz do skryptu. Zaczniemy od utworzenia tabelki w bazie danych:

```
CREATE TABLE `albumy` (
  `id` INT NOT NULL AUTO_INCREMENT PRIMARY KEY ,
  `tytul` VARCHAR( 50 ) NOT NULL ,
  `zespol` VARCHAR( 50 ) NOT NULL ,
  `rok` SMALLINT NOT NULL
) ENGINE = MYISAM;

INSERT INTO `albumy` VALUES (1, 'Nightflight to Venus', 'Boney M.', 1978);
INSERT INTO `albumy` VALUES (2, 'Take The Heat Off Me', 'Boney M.', 1976);
INSERT INTO `albumy` VALUES (3, 'Spirits Having Flown', 'Bee Gees', 1979);
INSERT INTO `albumy` VALUES (4, 'Saturday Night Fever', 'Bee Gees', 1977);
```

Kolejnym krokiem jest napisanie samego skryptu. Dane dla listy dostarczane są w postaci tablicy zawierającej poszczególne jej elementy. Są one również tablicami zawierającymi pojedyncze informacje, np. *tytul* czy *zespol*. Tak więc takie coś musimy generować.

```
<?php
define('OPT_DIR', './lib/');
require('./lib/opt.class.php');
```

```

try
{
    $tpl = new optClass;
    $tpl -> root = './templates/';
    $tpl -> compile = './templates_c/';
    $tpl -> gzipCompression = true;
    $tpl -> httpHeaders(OPT_HTML);

    $pdo = new PDO('mysql:host=localhost;port=3305;dbname=test',
        'root', 'root'); // 1

    $stmt = $pdo -> query('SELECT * FROM albumy ORDER BY rok');
    $result = array(); // 2

    while($row = $stmt -> fetch())
    {
        $result[] = array( // 3
            'tytul' => $row['tytul'],
            'zespol' => $row['zespol'],
            'rok' => $row['rok']
        );
    }
    $stmt -> closeCursor();

    $tpl -> assign('album', $result); // 4

    $tpl -> parse('szablon4.tpl');
}
catch(optException $exception)
{
    optErrorHandler($exception);
}
?>

```

Jego działanie jest następujące:

1. Najpierw łączymy się z MySQL-em za pomocą biblioteki PHP Data Objects.
2. Przygotowujemy pustą tablicę.
3. Zapisujemy tablicę z danymi o każdym albumie do naszego zbiornika na elementy: *\$result* jako kolejny jego element.
4. Korzystamy z metody *assign()*, aby przypisać wygenerowaną tablicę do naszej sekcji *album*.

To wszystko. Po uruchomieniu tego skryptu zobaczysz, że OPT wygenerował nam ładną listę albumów. Pójdźmy jednak dalej od strony szablonu, aby przekonać się nieco o możliwościach biblioteki. Na początek zastanówmy się, co będzie, jeśli w bazie albumów niczego nie będzie. Odpowiedź jest prosta: dostaniemy w kodzie puste znaczniki **<ul></ul>** i nic więcej. Istnieje kilka sposobów na powiadomienie internauty o kłopotcie - co ważniejsze, żaden z nich nie wymaga zmieniania czegokolwiek w kodzie PHP! Możemy sprawdzić instrukcją warunkową, czy sekcja zawiera jakieś elementy, a jeśli nie, powiadomić go o tym:

```

<html>
<head>
    <title>Open Power Template: albumy muzyczne</title>
</head>
<body>
    <h3>Moja lista albumów</h3>

    {if count($album) > 0}
    <ul>
    {section=album}
    <li>{$album.tytul} - {$album.zespol} ({$album.rok})</li>
    {/section}

```

```

    </ul>
    {else}
    <p>Brak albumów!</p>
{/if}
</body>
</html>

```

Jednak w tym przypadku zbliżamy się za bardzo do programowania. OPT udostępnia lepszy oraz nieco szybszy sposób polegający na rozbudowaniu sekcji o dodatkowe znaczniki: `{show}` oraz `{/show}`. Jeżeli zdecydujemy się na ich zastosowanie, to właśnie w nich ustalamy parametry sekcji, natomiast znacznik `{section}` pozostaje pusty:

```

<html>
<head>
  <title>Open Power Template: albumy muzyczne</title>
</head>
<body>
  <h3>Moja lista albumów</h3>

  {show=album}
  <ul>
    {section}
    <li>{$album.tytul} - {$album.zespol} ({$album.rok})</li>
    {/section}
  </ul>
  {showelse}
  <p>Brak albumów!</p>
{/show}
</body>
</html>

```

Teraz OPT ma jasno określone: to jest otoczenie listy będące jej integralną częścią; to jest wygląd pojedynczego elementu listy; to jest tekst alternatywny, jeśli lista nie posiada elementów.

Sekcja umożliwia wykonywanie prostych manipulacji na danych. Spróbujmy teraz wyświetlić ją w odwrotnej kolejności (nadal nie ruszając nic w kodzie PHP!) oraz jakoś wyróżnić pierwszy element:

```

<html>
<head>
  <title>Open Power Template: albumy muzyczne</title>
</head>
<body>
  <h3>Moja lista albumów</h3>

  {show=album; reversed}
  <ul>
    {section}
    {if $opt.section.album.first}
    <li><strong>{$album.tytul}</strong> -
      {$album.zespol} ({$album.rok})</li>
    {else}
    <li>{$album.tytul} - {$album.zespol} ({$album.rok})</li>
    {/if}
    {/section}
  </ul>
  {showelse}
  <p>Brak albumów!</p>
{/show}
</body>
</html>

```

Jeśli chodzi o odwracanie kolejności, wystarczy w parametrach sekcji dopisać słowo "reversed". Wyróżnienie pierwszego elementu to sprawa nieco bardziej skomplikowana. Musimy skorzystać koniecznie z pomocy instrukcji warunkowej, lecz OPT w tym momencie pomaga automatycznie stworzyć odpowiedni warunek.

Blok specjalny `$opt` daje dostęp do różnych ciekawych danych - w tym wypadku w postaci `$opt.section.album.first` zwraca on **true**, jeżeli aktualnie przetwarzany element sekcji `album` jest jej pierwszym elementem. Możemy z tego skorzystać do zdefiniowania dla niego alternatywnego wyświetlania.

## Wielopoziomowe listy

OPT zezwala na zagnieżdżanie sekcji, co umożliwia tworzenie wielopoziomowych list. Od strony szablonu sprawa jest oczywista - umieszczamy jedną sekcję w drugiej i nie martwimy się o resztę. Parę ważnych decyzji czeka nas natomiast od strony kodu PHP. Musimy zdecydować się na to, w jakim formacie dostarczymy dane do sekcji zagnieżdżonych. Mamy dwa wyjścia:

1. *Wiele sekcji - wiele tablic* - domyślne ustawienie OPT. Każda zagnieżdżona sekcja to osobna tablica, osobno przypisywana metodą `assign()`. Jedyną różnicą to większa liczba indeksów pozwalająca na skojarzenie elementów podrzędnych z nadrzędnymi. Przykład: jeśli kategoria 5 znajduje się w `$kategorie[5]`, to czwarty produkt leżący w tej kategorii znajdzie się w `$produkty[5][4]`.
2. *Wiele sekcji - jedna tablica* - dane sekcji podrzędnych trafiają do tablicy sekcji głównej. Posługując się przykładem z kategoriami i produktami, naszą kategorię 5 będziemy mieli po dawnemu w `$kategorie[5]` (zauważ, że w kwestii obsługi sekcji pojedynczych oba sposoby niczym się nie różnią!), natomiast produkt 4 będzie już w `$kategorie[5]['produkty'][4]`.

Wyboru najlepiej dokonać przed rozpoczęciem pisania aplikacji, choć warto też dodać, że gdy już się zdecydujemy, wcale nie jesteśmy skazani do samego końca na korzystanie z niego. Zawsze istnieje możliwość skorzystania z drugiego wariantu, lecz wymaga to poczynienia pewnych zmian po stronie szablonu.

Ale przejdźmy do rzeczy. Do demonstracji zagadnienia wykorzystamy tym razem tabele SQL dołączone do przykładów OPT. Odpowiedni plik znajduje się w katalogu `/examples/samples.sql` i jest dołączony do każdego wydania biblioteki. Po jego wgraniu pojawią nam się tabelki "products" oraz "categories" połączone prostą relacją jeden-do-wielu.

Poniżej podam teraz szablon z zagnieżdżonymi sekcjami, który umożliwi nam wyświetlenie naszych dwóch tabel. Następnie zaprezentowane zostaną dwa wykorzystujące go skrypty demonstrujące użycie każdego z formatów danych. Oczywiście nie będziemy w nich wykonywać zapytań rekurencyjnych; i tak nam nie pozwoli na to biblioteka PDO użyta do komunikacji z bazą. Z tego powodu trzeba jeszcze zadbać o mechanizm pamiętający, jaki ID kategorii powiązany jest z jakim indeksem tablicy `$kategorie`; nie możemy w tym celu wykorzystać tego samego ID, ponieważ OPT wymaga ciągłej numeracji zaczynającej się od 0, podczas gdy identyfikatory bazy danych nie spełniają ani jednego, ani drugiego warunku.

```
<html>
<head>
  <title>Open Power Template: lista produktów</title>
</head>
<body>
  <h3>Moja lista produktów</h3>

  <ul>
    {section=kategoria}
    <li><i>{$kategoria.nazwa}</i><br/>
    <table width="60%" border="1">
      <tr>
        <td width="30"><b>#</b></td>
        <td width="20%"><b>Nazwa</b></td>
        <td width="*"><b>Opis</b></td>
      </tr>
      {section=produkt}
      <tr>
        <td width="30">{$produkt.id}</td>
        <td width="20%">{$produkt.nazwa}</td>
        <td width="*">{$produkt.opis}</td>
      </tr>
    </li>
  </ul>
</body>
```

```

    {/section}
  </table>
</li>
{/section}
</ul>
</body>
</html>

```

Tak jak mówiłem, szablon nie jest szczytem wyrafinowania. Przyjrzyjmy się zatem pierwszemu skryptowi, który potrafi generować dane na jego użytek.

```

<?php
define('OPT_DIR', './lib/');
require('./lib/opt.class.php');

try
{
    $tpl = new optClass;
    $tpl -> root = './templates/';
    $tpl -> compile = './templates_c/';
    $tpl -> gzipCompression = true;
    $tpl -> httpHeaders(OPT_HTML);

    $pdo = new PDO('mysql:host=localhost;port=3305;dbname=test',
        'root', 'root');

    $stmt = $pdo -> query('SELECT id, name FROM
        categories ORDER BY id');
    $categories = array();
    $categoryMatch = array();
    $i = 0;
    while($row = $stmt -> fetch())
    {
        $categories[$i] = array(
            'id' => $row['id'],
            'nazwa' => $row['name']
        );
        $categoryMatch[$row['id']] = $i; // 1
        $i++;
    }
    $stmt -> closeCursor();
    unset($stmt);

    $stmt = $pdo -> query('SELECT id, name, description,
        category FROM products ORDER BY category, id');
    $products = array();

    while($row = $stmt -> fetch())
    {
        $products[$categoryMatch[$row['category']]][] = array( // 2
            'id' => $row['id'],
            'nazwa' => $row['name'],
            'opis' => $row['description']
        );
    }
    $stmt -> closeCursor();

    $tpl -> assign('kategoria', $categories); // 3
    $tpl -> assign('produkt', $products);

    $tpl -> parse('szablon5.tpl');
}

```

```

    catch(optException $exception)
    {
        optErrorHandler($exception);
    }
?>

```

Oto opis jego działania:

1. *\$categoryMatch* jest tablicą pamiętającą, jaki ID kategorii znajduje się w którym elemencie listy. Będziemy ją potem wykorzystywać, aby połączyć produkty z kategoriami.
2. Tutaj musimy skonstruować nieco większą tablicę produktów uwzględniającą fakt, iż te są zagnieżdżone wewnątrz kategorii. Posiada ona dwa indeksy - pierwszy pokazuje, do jakiej kategorii należy produkt (tu wstawiamy wynik z tablicy *\$categoryMatch*), a drugi jest indeksem produktu w obrębie tej kategorii.
3. Do szablonu przekazujemy dwie tablice, osobno dla każdej z sekcji.

Jak wspomniałem, jest to domyślny sposób przekazywania danych do sekcji zagnieżdżonych, lecz OPT posiada jeszcze jeden. Należy go ręcznie włączyć, ustawiając dyrektywę *sectionStructure* na *OPT\_SECTION\_SINGLE*. Wtedy zamiast dwóch tablic, będziemy generowali tylko jedną - dla sekcji nadrzędnej. Dane sekcji podrzędnych zostaną w niej umieszczone jako poszczególne bloki. Od strony szablonu nic się nie zmienia. Od strony PHP, musimy zmienić trzy linijki powyższego skryptu. Na początek dodaj w części inicjacyjnej informację, że używamy innej struktury sekcji:

```
$tpl -> sectionStructure = OPT_SECTION_SINGLE;
```

Następnie w pobieraniu produktów zamień linijkę:

```
$products[$categoryMatch[$row['category']]][ ] = array(
```

na:

```
$categories[$categoryMatch[$row['category']]][ 'produkt' ][ ] = array(
```

Zauważ, że teraz w obrębie kategorii tworzymy blok "produkt", lecz zamiast tekstu czy liczby, umieszczamy w nim dane dla sekcji **produkt** powiązane z aktualnie przetwarzaną kategorią.

Ponieważ nie tworzymy już tabeli *\$products*, poniższa linijka jest nam zbędna i należy ją usunąć:

```
$tpl -> assign('produkt', $products);
```

Usuń z *templates\_c* skompilowaną wersję dotychczasowego szablonu i odpal zmodyfikowany skrypt. To wszystko, właśnie poznałeś dwie metody tworzenia zagnieżdżonych sekcji. Pamiętaj, że powinieneś na początku prac zdecydować, którego z nich będziesz używać, jednak jeżeli nagle przyjdzie Ci użyć drugiego, nie panikuj - wprawdzie wymaga to pewnych modyfikacji po stronie szablonów, ale jest wykonalne. Sekcje posiadają dodatkowy parametr, *datasource*, za pomocą których możesz powiadomić kompilator, gdzie dokładnie znajdują się dane dla nich. I tak: jeżeli korzystasz z pierwszego sposobu, a potrzebny Ci jest drugi, powinieneś połączyć sekcję produktów z kategoriami w następujący sposób:

```
{section name="produkt" datasource="$kategoria.produkt"}
```

W przeciwną stronę, musimy zrobić tak:

```
{section name="produkt" datasource="$produkt"}
```

Osobną sprawą jest wyświetlanie zawartości drzew z użyciem OPT. W wersjach 1.0.x wymaga to sporego nakładu pracy i wyposażenia sekcji w cały zestaw instrukcji warunkowych, lecz podczawszy od wersji 1.1.x biblioteka ma już posiadać specjalne instrukcje realizujące to zadanie schludnie i elegancko. Pokażę zatem tutaj sposób uzyskania drzewka dla wersji 1.0.x. Zakładam, że generujesz je algorytmem *modified preorder tree traversal*, który generuje liniową listę elementów z dołączonym do nich parametrem *depth* określającym ich głębokość. Zanim prześlemy takie drzewo do OPT, musimy przepuścić je przez dodatkową funkcję:

```

function prepareTree($tree)
{
    foreach($tree as $id => &$item)
    {
        if($item['depth'] == @$tree[$id+1]['depth'])
        {
            $item['leaf'] = 1;
        }
    }
}

```

```

    }
    if($item['depth'] < @$tree[$id+1]['depth'])
    {
        $item['opening'] = 1;
    }
    if($item['depth'] > @$tree[$id+1]['depth'])
    {
        $item['closing'] = 1;
        if(isset($tree[$id+1]))
        {
            $item['toclose'] = ($item['depth']
                - @$tree[$id+1]['depth']);
        }
    }
}
return $tree;
} // end prepareTree();

```

Umieści ona w każdym elemencie drzewa następujące bloki:

1. *leaf* - podany element jest liściem drzewa
2. *opening* - po aktualnym elemencie należy otworzyć nową listę dla jego dzieci.
3. *closing* - dany element jest ostatnim na danym poziomie.
4. *toclose* - umieszczany razem z *closing*. Informuje, ile list należy teraz zamknąć, aby powrócić do poziomu następnego elementu na liście.

Teraz przechodzimy do strony szablonu. Tutaj będziemy musieli pobawić się nieco instrukcjami warunkowymi i pętlami, aby algorytm zadziałał prawidłowo. Oto i on:

```

<ol>
  {section=tree}
    {if $tree.leaf} {* 1 *}
    <li>{$tree.title}</li>
    {/if}
    {if $tree.opening} {* 2 *}
    <li>{$tree.title}<ol>
    {/if}
    {if $tree.closing} {* 3 *}
    <li>{$tree.title}</a>
      {for=@i is 0; @i < $tree.toclose; @i is @i+1} {* 4 *}
      </li></ol>
    {/for}
  </li>
  {/if}
{/section}
</ol>

```

Znaczenie poszczególnych fragmentów:

1. Wygląd liścia drzewa.
2. Wygląd elementu otwierającego nowy poziom listy.
3. Wygląd elementu zamykającego aktualny poziom listy.
4. Pętla zamykająca listy do nowego poziomu.

Teraz jesteśmy już w stanie wyświetlać struktury drzewiaste za pomocą Open Power Template.

## Interfejs wielojęzyczny

Przejdźmy teraz do nowego zagadnienia. Przypuśćmy, że tworzysz duży projekt skierowany nie tylko na rynek polski, ale i klientów zagranicznych. Niestety nasz rodzimy język jest dopiero na początku realizacji

planu zawładnięcia światem, dlatego lepiej jest założyć, że taki Brytyjczyk lub Niemiec nie potrafi się nim jeszcze posługiwać. Wynika z tego, iż nasza witryna musi posiadać wielojęzyczny interfejs: wszystkie komunikaty i napisy zmieniają się w zależności od tego, jaką wersję językową wybrał sobie internauta. Do normalnego systemu szablonów musielibyśmy przekazywać tony informacji lub wykorzystywać w szablonach bezpośrednio elementy silnika PHP witryny, aby nieco ułatwić ten proces. OPT normalnym systemem szablonów jednak nie jest i posiada specjalny mechanizm ułatwiający tworzenie wielojęzycznych interfejsów. Tradycyjnie, zacznijmy jednak od szablonu:

```
<html>
<head>
  <title>Open Power Template: i18n</title>
</head>
<body>
<p>{ $page1@intro}</p>
<p>{ $page1@preview}</p>
<ul>
  <li>{literal}{ $global@yes}</literal>: { $global@yes}</li>
  <li>{literal}{ $global@no}</literal>: { $global@no}</li>
  <li>{literal}{ $global@maybe}</literal>: { $global@maybe}</li>
</ul>
{apply( $page1@counter, 1 )}
<p>{ $page1@counter}</p>
{apply( $page1@counter, 2 )}
<p>{ $page1@counter}</p>
</body>
</html>
```

W kodzie poumieszczane są nowe, niespotykane dotąd rodzaje bloków mające postać **{ \$grupa@nazwa }**. Są to tzw. bloki językowe, które zaznaczają miejsca, w których mają pojawić się jakieś komunikaty interfejsu. OPT, napotykając taki blok, odczytuje sobie z niego ID komunikatu oraz grupy, do której należy, po czym samodzielnie pobiera go sobie ze zdefiniowanego wcześniej systemu i18n. Dodam jeszcze, że użyta w szablonie instrukcja **{literal}** powoduje wyświetlanie znajdujących się wewnątrz niej tagów OPT, zamiast ich przetwarzania. Zastosowałem ją tutaj, aby wyświetlić nazwy wykorzystywanych bloków językowych.

Kolejnym ciekawym elementem jest funkcja *apply()*. Jej zadanie można wytłumaczyć następująco: czasami niektóre dane muszą być umieszczone wewnątrz komunikatu, co ma miejsce np. w takim przypadku: "Na stronie jest 17 użytkowników on-line". Widzimy, że "17" jest wartością dynamiczną, która musi być zassana z jakiegoś bloku. Mamy dwa wyjścia: albo rozbijemy komunikat na dwa, np. *\$stats@users\_online1* i *\$stats@users\_online2*, a między nimi umieścimy blok, lub... zastosujemy funkcję *apply()*, która odnajdzie w tekście specjalny znacznik (*%s*) i wstawi w jego miejsce jakąś wartość dynamiczną. Właśnie tak robimy powyżej: mamy dynamiczny blok językowy *\$page@counter*, który ma treść "Licznik: %s.". Odpowiednią liczbę umieszczamy na miejscu *%s* właśnie za pomocą funkcji *apply()*.

Aby używać wielojęzycznych interfejsów, musimy w kodzie PHP połączyć OPT z jednym z nich. Biblioteka zezwala na wykorzystanie dwóch rodzajów interfejsu:

1. Proceduralny - do OPT podpinamy dwuwymiarową tablicę asocjacyjną wiążącą ID komunikatów z ich treścią.
2. Obiektowy - do OPT podpinamy obiekt klasy implementującej interfejs *ioptI18n* - klasa ta ma za zadanie udostępniać żądane komunikaty za pomocą odpowiednich metod. Oczywiście daje nam to znacznie większe pole do popisu - możemy wtedy zaprogramować np. automatyczne wczytywanie grup komunikatów, gdy są używane po raz pierwszy.

Zajmiemy się najpierw pierwszym z nich. Skrypt jest tutaj niezwykle prosty: tworzymy sobie tablicę z wszystkimi tekstami, po czym podpinamy ją do OPT metodą *optClass::setDefaultI18n()*:

```
<?php
define('OPT_DIR', './lib/');
require('./lib/opt.class.php');

$daneJezykowe = array(
  'global' => array(
```

```

        'yes' => 'Tak',
        'no' => 'Nie',
        'maybe' => 'Być może'
    ),
    'page1' => array(
        'intro' => 'Witaj w testowym projekcie opartym
            o Open Power Template!',
        'preview' => 'Podgląd możliwości i18n',
        'counter' => 'Licznik: %s'
    )
);

try
{
    $tpl = new optClass;
    $tpl -> root = './templates/';
    $tpl -> compile = './templates_c/';
    $tpl -> gzipCompression = true;
    $tpl -> httpHeaders(OPT_HTML);

    $tpl -> setDefaultI18n($daneJezykowe);

    $tpl -> parse('szablon3.tpl');
}
catch(optException $exception)
{
    optErrorHandler($exception);
}
?>

```

Uruchamiamy skrypt i niby wszystko działa, ale jednak coś jest nie tak. Spójrzmy na nasz licznik. Drugie wywołanie funkcji *apply()* wcale nie zaktualizowało licznika! Powód jest prosty - rezultat jej działania nadpisał oryginalny tekst, kasując znacznik. W końcu gdzieś musiał on trafić, a skoro do dyspozycji jest tylko jedna tablica...

Jest jednak sposób, aby temu zaradzić, a mianowicie napisać własny, obiektowy interfejs i18n.

```

<?php
define('OPT_DIR', './lib/');
require('./lib/opt.class.php');

class i18n implements ioptI18n
{
    private $tpl;
    private $replacements = array();
    private $data = array(
        'global' => array(
            'yes' => 'Tak',
            'no' => 'Nie',
            'maybe' => 'Być może'
        ),
        'page1' => array(
            'intro' => 'Witaj w testowym projekcie opartym
                o Open Power Template!',
            'preview' => 'Podgląd możliwości i18n',
            'counter' => 'Licznik: %s'
        )
    );

    static private $instance;

    private function __construct(){ }
}

```

```

public function setOptInstance(optClass $tpl)
{
    $this -> tpl = $tpl;
} // end setOptInstance();

static public function getInstance()
{
    if(!is_object(self::$instance))
    {
        self::$instance = new i18n;
    }
    return self::$instance;
} // end getInstance();

public function put($group, $text_id)
{
    if(isset($this -> replacements[$group][$text_id]))
    {
        return $this -> replacements[$group][$text_id];
    }
    return $this -> data[$group][$text_id];
} // end put();

public function apply($group, $text_id)
{
    $args = func_get_args();
    unset($args[0]);
    unset($args[1]);
    $this -> replacements[$group][$text_id]
        = vsprintf($this -> data[$group][$text_id], $args);
} // end apply();

public function putApply($group, $text_id)
{
    $args = func_get_args();
    unset($args[0]);
    unset($args[1]);
    return vsprintf($this -> data[$group][$text_id], $args);
} // end putApply();
}

try
{
    $tpl = new optClass;
    $tpl -> root = './templates/';
    $tpl -> compile = './templates_c/';
    $tpl -> gzipCompression = true;
    $tpl -> httpHeaders(OPT_HTML);

    $tpl -> setObjectI18n(i18n::getInstance());

    $tpl -> parse('szablon3.tpl');
}
catch(optException $exception)
{
    optErrorHandler($exception);
}

?>

```

Przyjrzyjmy się dokładniej klasie *i18n*. Po interfejsie *iopt18n* implementuje ona następujące metody:

1. **put(\$group, \$text\_id)** - metoda zwraca treść komunikatu *\$text\_id* w grupie *\$group*.
2. **apply(\$group, \$text\_id, ...)** - nasza własna implementacja funkcji *apply()*. Pobiera ona zmienną liczbę parametrów - pierwsze dwa to oczywiście nazwa grupy oraz ID tekstu, natomiast dalsze to wartości do umieszczenia w podanym bloku językowym. Znaczniki *%s* najlepiej podmieniać funkcją *vsprintf()*, która ma wszystko, co nam jest potrzebne.
3. **putApply(\$group, \$text\_id, ...)** - ta metoda jest wprowadzona dla wygody programisty, który może interfejs wykorzystywać także poza OPT. Działa ona tak, jak *apply()*, lecz zamiast zapisywać rezultat do jakiegoś bufora, zwraca go.
4. **setOptInstance(optClass \$tpl)** - pobiera obiekt klasy *optClass*.

Podana powyżej implementacja wykorzystuje wzorec singleton, aby mieć pewność, że po skrypcie nie będzie się pałętać 10 obiektów interfejsu językowego. Oryginalne komunikaty zapisane są w tablicy *\$replacements*, a teksty zmodyfikowane przez *apply()* trafiają do tablicy *\$data* - dzięki takiemu podziałowi nasza klasa prawidłowo obsłuży podany wyżej szablon; oryginalny komunikat źródłowy nigdy nie zostanie nadpisany.

## Koniec cz. 1

Poznaliśmy właśnie podstawowe zasady i mechanizmy dostępne w OPT. Część druga tego artykułu jest w całości poświęcona systemowi komponentów. Nie tylko nauczymy się z nich korzystać, ale też napiszemy własny. Zapraszam do lektury części drugiej.

**Tomasz "Zyx" Jędrzejewski**

Aktualna wersja artykułu zawsze na **[www.zyxist.com](http://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**