PHP posiada w najnowszych wersjach (od 5.3.3) wbudowany mechanizm przestrzeni nazw . Wcześniej takiego mechanizmu nie było - lub można uznać, że była jedna wspólna przestrzeń nazw. Jednak wykorzystując zalety OO [ang. Object Oriented] PHP5 można uzyskać bardzo podobny efekt do rzeczywistych przestrzeni nazw. Jest to konwencja zaczerpnięta z repozytorium
PEAR
.

Na początek

- choćby podstawowa wiedza nt. OO PHP5
- umiejętność ładowania skryptów za pomocą autoloadera
- serwer www (może być lokalny np. WAMP)
- jakieś 10-15 minut

Prosta idea

W językach, które opierają się na przestrzeniach nazw (pakietach, modułach - zwał, jak zwał), np. Java pakiety są pogrupowane w katalogi. I tak, kiedy mamy taką strukturę katalogów:

{codecitation}

/java /util /Data.java /sql /Connector.java

{/codecitation}

W takiej sytucji do klasy Data mielibyśmy taką ścieżkę:

{codecitation}java.util.Data zmienna = new java.util.Data();{/codecitation}

Oczywiście można użyć tam mechanizmu importów, pozwalających na wykluczenie java.util. Chciałem jednak pokazać, jaka jest zasada w językach "dojrzalszych" obiektowo niż PHP5.

PHP5 i konflikty nazw

W PHP5 możemy uzyskać bardzo podobny efekt. Załóżmy, że mamy taką strukturę katalogów ("pakietów"):

{codecitation}

/Validation /Type.php /Value.php /Exception /Type.php /Value.php

{/codecitation}

Jeśli chcemy zatem załączyć odpowiednie pliki stosujemy:

{codecitation class='brush:php'}

require_once 'Validation/Exception/Type.php'; require_once 'Validation/Type.php'; $oValidator = new Type(); // tworze obiekt walidujacy typy zmiennych

{/codecitation}

Niestety w PHP sama struktura katalogów nie tworzy przestrzeni nazw. Tak więc klasy z plików `Validation/Type.php' i `Validation/Exception/Type.php' nazywają się `Type'. Co powoduje konflikt nazw. Dodatkowo przy załączaniu plików mamy wiele dodatkowych nic-nie-robiących linii kodu. A można temu zapobiec, przy odpowiedniej konwencji.

Konwecja PEAR

Skoro mamy już odpowiednią strukturę katalogów, to może użyć metody znanej np. z Javy czy C#: Poniższy kod nie będzie działał, ale ma pokazać do czego dążymy:

{codecitation class='brush: php'}$oValidator = new Validation.Type();{/codecitation}

Jak już powiedziałem, to jest kod, który nie działa w PHP. Po prostu PHP nie posiada operatora kropki, który miałby tu wykonać upragnioną przez nas funkcjonalność. Dlaczego jednak nie wprowadzić znaku podkreślenia, zamiaast kropki? Wtedy kod wyglądałby tak:

{codecitation class='brush: php'}$oValidator = new Validation_Type();{/codecitation}

W takiej sytuacji należy jeszcze zmienić definicję klasy:

{codecitation class='brush: php'}class Type { ... }{/codecitation}

na

{codecitation class='brush: php'}class Validation_Type { ... }{/codecitation}

Przy załączaniu robimy identycznie jak poprzednio:

{codecitation class='brush:php'}

require_once 'Validation/Exception/Type.php'; require_once 'Validation/Type.php'; $oValidator = new Validation_Type(); // tworze obiekt walidujacy typy zmiennych

{/codecitation}

I już po konflikcie nazw. Klasa w pliku `Validation/Exception/Type.php' nazywa się `Validation_Exception_Type', a z pliku `Validation/Type.php' - `Validation_Type'. Co prawda trochę nam się wydłużył kod, jednak uzyskaliśmy bardzo porządaną cechę kodu - modularność. Co nam po wcześniejszej "modularności" jeśli mieliśmy klasę `User' w &quot;module&quot; forum i klasę `User' w module "Newsletter". Wtedy obie miały taką samą nazwę, teraz `Forum_User' oraz 'Newsletter_User'... Oczywiście, to rozwiązanie nadal nie jest idealne - nadal konflikt może się zdarzyć. Jest jednak mniej prawdopodobny.

Automatyczne załączanie klas

Skoro w nazwie skryptu, definiujemy jego położenie, to dlaczego nie wykorzystać odpowiednio autoloadera ?

{codecitation class='brush: php'}

function __autoload($name) { $dirs = explode('_', $name); $path = implode('/', $dirs) . '.php'; // echo $path . '<hr />'; odkomentuj ta linie, jesli chcesz zobaczyc co sie laduje if (file_exists($path)) { require_once $path; } else { die('blad'); // przed zabiciem skryptu, mozesz wyslac sobie informacje o bledzie } } $oValidator = new Validation_Type();

{/codecitation}

I wszystko działa. Wszystkie dziwne rzeczy dzieją się w tle. Bardzo podobny system wykorzystywany jest we frameworku Zend oraz kilku innych frameworkach. Powodzenia.

Pełne kody na SVN

http://php-validation.googlecode.com/svn/trunk/