Něco o pclib verze 2.x

Úvod

V následujících příkladech budeme předpokládat, že máme vytvořenou pclib aplikaci s připojením do databáze, protože většina komponent pracuje s databází.

Inicializace databáze

To zajistíme pomocí následující řádků:

```
require "libs/pclib/pclib.php";

$app = new pclib\App('demo-app');
$app->db = new pclib\Db('pdo_mysql://user:heslo@localhost/demo-app-databaze');

//(...a zde bude kod nasledujicich prikladu)
```

Tento kousek kódu vytvoří připojení k databázi pomocí PDO, což je moderní php rozhraní pro jednotný přístup k databázovým systémům, nahrazující staré funkce mysql_xxx(), pg_xxx(), ...

Vytvoření tabulek

Potřebné databázové tabulky můžete vytvořit tak, že si stáhnete nástroj **padmin**, nahrajete ho do adresáře **demo-app/padmin** a nastavíte databázové připojení v souboru **demo-app/padmin/config.php**

Nyní zadejte v prohlížeči **localhost/demo-app/padmin/?r=install**

*Nebo můžete spustit skript **pclib_mysql.sql** nacházející se v instalačním balíčku v adresáři **pclib/install**.*

Konvence zápisu kódu

Pclib 2 používají styl zápisu kódu **camelCase**, který je používán ve všech moderních php frameworcích a knihovnách.

Ukázka:

```
$nazevObjektu = new NazevTridy;
$nazevObjektu->nazevMetody($nazevPromenne, NAZEV_KONSTANTY);
```

Translator

K překladu textu slouží třída Translator. Používá se tak, že načteme texty k překladu a následně zavoláme funkci translate(), která zobrazí zvolenou jazykovou verzi.

Texty k překladu mohou být uloženy buď v databázi, nebo v souboru a oba způsoby je možné i kombinovat.

Texty v souboru

```
//--- soubor languages/en.php ---

$messages = array
(
      'Ahoj světe!' => 'Hello world!',
      'Dnes je %s' => 'Today is %s',
      'localized-date-format' => 'm/d/Y',
);

//--- soubor index.php ---

$tr = new Translator;
$tr->language = 'en';
$tr->useFile('languages/en.php');

print $tr->translate('Ahoj světe!');           //-> Hello world!
print $tr->translate('Nepřeložený text');      //->  Nepřeložený text

$date = date($tr->translate('localized-date-format'));
print $tr->translate('Dnes je %s', [$date]);   //-> Today is 12/9/2016
```

Texty v databázi

```
$tr = new Translator;
$tr->language = 'en';
$tr->usePage('products');

print $tr->translate('Seznam produktů'); //-> List of products
```

Funkce usePage() načte z databáze texty pro stránku 'products'. Je to proto, aby se do paměti nemusely načítat naráz všechny texty, které aplikace používá. Můžete načíst i několik stránek současně.
Při dnešních velikostech pamětí je ale často nejpohodlnější řešení mít všechny texty uložené v jediné stránce pojmenované například 'default'.

Nastavení jazyka aplikace

Třída App slouží jako fasáda zjednodušující přístup k objektům, které by bylo jinak nutné vytvářet a konfigurovat ručně. Toto je dobrý příklad - nastavení jazyka lze provést i takto:

```
$app->language = 'en';
```

Tento řádek zavolá metodu $app->setLanguage(), která vytvoří objekt Translator a přiřadí ho proměnné $app->translator, kde je přístupný všem komponentám frameworku.

Zároveň načte hlášení a popisky pclib ze souboru pclib/localization/xx.php (pokud existuje) a načte stránku 'default' z databáze textů.

Jestliže chcete jen nastavit popisky pclib do češtiny a nechcete načítat nic z databáze, můžete použít příkaz

```
$app->setLanguage('en', false);
```

(Parametr false vypne načítání z databáze)

Pokud potřebujete naplnit databázi textů, stačí nastavit

```
$app->language = 'source';
```

a proklikat aplikaci. Vypisované texty se automaticky přidají do tabulky a lze je spravovat v nástroji padmin.

Logger

Uložení hlášení do databázového logu provedeme takto:

```
$logger = new Logger;
$logger->log('info', 'product/delete', null, $product_id);
```

- První parametr je kategorie, může být libovolná, lze podle ní vyhledávat.
  Pclib používají např. AUTH_ERROR, Error, Warning pro chyby php.
- Druhý parametr je zpráva - logger ukládá v rámci úspory místa jen číselné id zprávy. Pokud neexistuje, přidá ji do číselníku a uloží opět její id. Nepoužívejte proto zprávy s proměnným obsahem například se jménem souboru nebo id objektu.
- K uložení id slouží čtvrtý parametr, zde ukládáme číslo produktu, který byl smazán.
- Třetí parametr, obvykle null, může obsahovat delší zprávu, která se uloží do zvláštní tabulky.

Vrátí posledních 100 řádků logu:

```
$array = $logger->getLog(100);
```

K vyhledávání a prohlížení logu lze použít padmin.

Nastavením aplikačního logu:

```
$app->logger = new Logger;
```

umožníte logovat i ostatním komponentám pclib a můžete používat funkci $app->log(), která ve výchozím stavu nedělá nic. Tak lze snadno vypínat a zapínat logování.

Layout

Layout je šablona, která obaluje všechny stránky aplikace a umožňuje využívat některé speciální elementy. Ukázka:

```
$app->setLayout('tpl/layout.tpl');
//...kod
$app->out(); //vypis obsahu s layoutem

//--- soubor tpl/layout.tpl ---

<?elements
messages PRECONTENT
navigator NAVIG
head HEAD scripts "css/bootstrap.css,js/jquery.js"
?>

<!DOCTYPE html>
<html>
  <head>{HEAD}</head>
      <body>{NAVIG} {PRECONTENT} {CONTENT}</body>
</html>

// --- end ---
```

Messages

Element **messages** určuje místo, kam se budou vypisovat takzvané flash-messages. Jsou to zprávy, které se mohou pamatovat i přes několik stránek a přesměrování (ukládají se v session) a vymažou se až v okamžiku, kdy se poprvé zobrazí.

```
$app->message('Formulář byl uložen.');
$app->redirect('products');
```

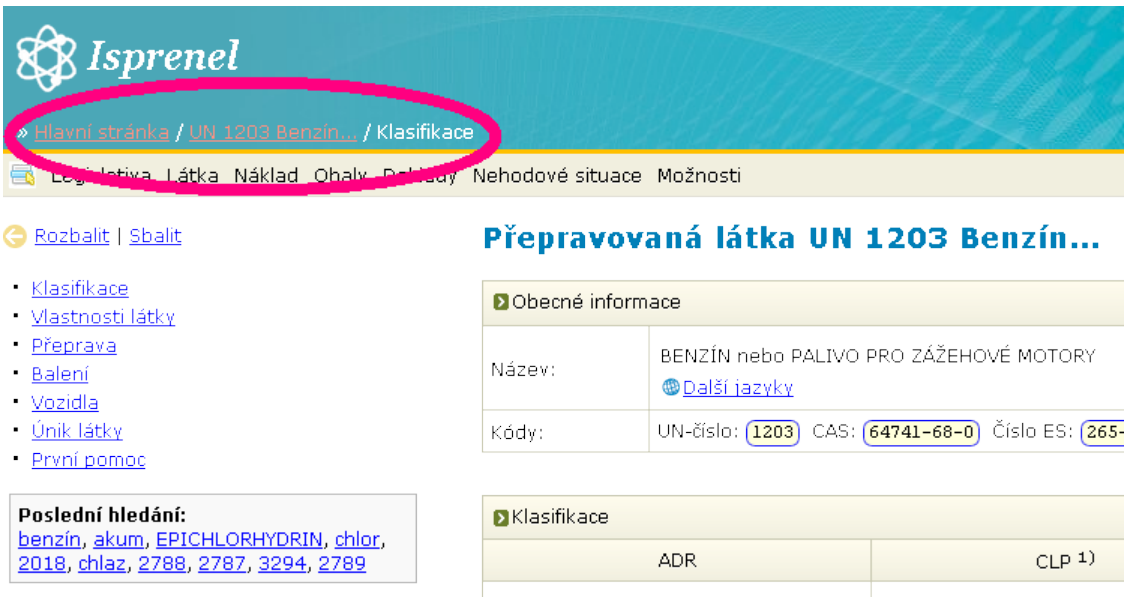
Zpráva se vypíše po přesměrování na následně zobrazené stránce.
Můžete si nastavit vlastní css-třídu pro div ve kterém se zpráva objeví:

```
$app->message('Pozor, toto je nebezpečné!', 'alert alert-danger');
```

Zprávy jsou také zařazeny mezi texty k překladu.

Navigator

Element **navigator** zobrazí tzv. breadcrumb navigator, často používaný navigační prvek.



Odkaz na aktuální stránku do něj přidáte zavoláním:

```
$app->layout->bookmark(1, 'Popisek');
```

(Uloží odkaz na aktuální url jako první položku navigátoru. 2 jako druhou atd.)

Head

Slouží k vygenerování odkazů na js a css skripty, pochopitelně s mnoha vylepšeními :)

```
// --- výstup ---
<link rel="stylesheet" type="text/css" href="/demo-app/css/bootstrap.css?
v=1401177354">
<script language="JavaScript" src="/demo-app/js/jquery.js?
v=1409126212"></script>
```

- Vytvoří absolutní cestu k souboru, což je důležité pro friendly-url
- V případě překlepu a linkování neexistujícího souboru ohlásí chybu, takže se o problému hned dozvíte
- Přidá verzi souboru. Tato verze se změní jen tehdy, pokud se změní obsah souboru, takže prohlížeč vrací soubory z cache, ale ve chvíli kdy je změníte, dojde k jejich znovunačtení a všichni uživatelé obdrží jejich aktuální verzi.

Skripty lze do hlavičky přidávat také dynamicky z php kódu. Například chceme přidat jquery plugin na zobrazení kalendářového pole jen na stránkách s formulářem, napíšeme:

```
$app->layout->addScripts('js/date-picker.js');
```

Auth

Přihlášení, odhlášení, práce s uživateli, čtení rolí, ověřování oprávnění - to je práce pro objekt Auth.

Jeho chování řídí následující konfigurační parametry:

```
'pclib.auth' => array('algo' => 'md5', 'secret' => 'write any random string!',
'realm' => ''),
```

Parametr **'algo'** udává algoritmus hashování hesla. Možnosti jsou md5 nebo bcrypt.
Doporučená volba je bcrypt, což použije php funkci **password_hash()** která je přímo určená ke kryptování hesla bezpečnou metodou.

Tato funkce je dostupná až od php 5.5, ale existuje polyfill knihovna která ji doplní i do starších verzí php.

Volba 'md5' je defaultní nastavení a je kompatibilní se staršími verzemi pclib. Při jeho použití nastavte **'secret'** na nějaký náhodný řetězec, který bude využit k posílení hesla.

Chcete-li aby se uživatel přihlásil do celé skupiny aplikací na stejném serveru, nastavte všem stejný parametr **'realm'** - např. 'stkportal'. Aplikace pak budou sdílet autentizační session. Implicitně je realm nastavený na jméno aplikace.

```
//prihlaseni uzivatele
$auth = new Auth;
print $auth->loggedUser;      //-> null   /lze též $auth->getUser();/
$auth->login('kovar', 'tajne'); //-> true
print $auth->loggedUser;      //-> Object.pclib\AuthUser
print $auth->loggedUser->values['FULLNAME']; //-> Jan Kovář
```

Pomocí funkce **login($jmeno, $heslo)** autorizace ověří uživatele a v případě že heslo souhlasí, přihlásí ho a přiřadí objekt AuthUser do **$auth->loggedUser**.

Za normálních okolností přiřadíme Auth do proměnné **$app->auth** aby byl dostupný i jiným částem frameworku.

Další funkce:

```
//získání jiného než přihlášeného uživatele:
$user = $auth->getUser('boss');
```

Pozor! Funkce getUser() vracela ve starších verzích pole s uživatelskými údaji, nikoli objekt!

Ověříme jestli má získaný uživatel heslo 'tajne':

```
$user->passwordVerify('tajne');     //-> false
print $user->values['roles'];       //-> ['admin', 'editor']
```

Stejně ho programově přihlásíme:

```
$auth->setLoggedUser($user);
$user->isLogged();      //-> true
```

Další funkce viz referenční příručka.

Obsluha chyb

Obsluhu chyb obstarává třída **pclib\system\ErrorHandler** a je řízena tímto konfiguračním parametrem:

```
'pclib.errors' => array('display', 'develop', /*,'log','template'=>'error.tpl' */),
```

Tento parametr by se měl lišit ve vývojovém, resp. produkčním prostředí.
V ostrém provozu nechceme vypisovat podrobná chybová hlášení s sql dotazů a trasováním volání funkcí, neboť prozrazují stavbu aplikace a jsou pomocí pro hackera, popřípadě mohou dokonce obsahovat bezpečnostně kritické údaje jako jméno a heslo pro přístup do databáze.
Naopak chceme chyby logovat, abychom mohli vypátrat jejich příčinu.

```
//nastaveni pro produkcni prostredi

'pclib.errors' => array('log'),

//nebo

'pclib.errors' => array('display', 'log'),
```

V prvním případě se chyby nezobrazují vůbec a pouze se logují, ve druhém se zobrazí, ale bez developerských informací - ty zapíná volba **'develop'**.
V tomto případě se pro zobrazení použije šablona **pclib/assets/error.tpl**, nebo kterákoli jiná šablona, kterou uvedete v parametru 'template'. Zároveň se odešle hlavička s kódem **500 Internal Server Error**.

BaseObject

Většina tříd pclib je odvozená od třídy **pclib\system\BaseObject**, která poskytuje určité základní funkce jako nastavení default parametrů, události a kontrolu existence atributu.

Tuto třídu můžete použít také vy, jako předka vašich objektů.

Kontrola existence atributu

PHP umožňuje přiřadit hodnotu do neexistujícího atributu objektu, což je obvykle chyba. BaseObject v takovém případě vyhodí výjjimku.

```
use pclib\system\BaseObject

$a = new stdClass;
$a->xxx = 'xxx'; //-> projde

class Konvertor extends BaseObject {
      public $values;
}

$b = new Konvertor;
$b->vaules = [];  //-> Cannot write to an undeclared property Konvertor->vaules.
```

Události

```
class Konvertor extends BaseObject {
      public $values;
      public $outputFormat = 'Html';

      public $onBeforeConvert; //udalosti jsou atributy s nazvem ve tvaru onXxx
      public $onAfterConvert;

function convert() {
      $this->onBeforeConvert(); //ale muzete je v objektu zavolat jako funkci
      $result = $this->doSomething();
      $this->onAfterConvert($result);
}

//... code ...

}

//Objekt vyvolavajici udalosti mame hotovy a nyni mu muzeme do udalosti neco
priradit:

$konvertor = new Konvertor;

$konvertor->onAfterConvert[] = function($event) {
      print 'Konverze skončila s výsledkem'. $event->data[0];
}

$konvertor->convert();
```

Default parametry

Pro libovolnou třídu (potomka BaseObject) můžete nastavit jaké má mít přednastavené hodnoty svých veřejných atributů.

```
Konvertor::defaults('outputFormat', 'Xml');
$x = new Konvertor;
print $x->outputFormat; //-> Xml
```

Využití systémových událostí pclib

Můžeme si například nechat zasílat případné php chyby mailem:

```
$app->errorHandler->onException[] = function($event) {
      $e = $event->data[0];
      mail('admin@super.com', 'Chyba v demo-app',  $e->getMessage());
};
```

Nebo logovat pomalé dotazy:

```
$slowQueryLog = function($event) use ($logger) {
      static $time;
      if ($event->name == 'onBeforeQuery') {
      $time = time();
}
else {
      $sql = $event->data[0];
      if (time()-$time > 2) $logger->log('warning', 'slowquery', $sql);
}
};

$app->db->onBeforeQuery[] = $slowQueryLog;
$app->db->onAfterQuery[] = $slowQueryLog;
```

Controllers - zastavení první

Každá webová aplikace se musí na základě url požadavku rozhodnout jaký kód se zavolá. Má proto smysl vyřešit tento proces směrování (routování) jednotně ve frameworku.

Toto směrování provádí v pclib metoda **$app->run()**.

Příklad:

Zadáme: **index.php?r=products/edit?id=100**

$app->run() vyhledá soubor **controllers/ProductsController.php**, který musí obsahovat třídu **ProductsController**. Tento objekt vytvoří a zavolá jeho metodu **editAction()**.

```
//$app->run() zjednodušeně:

require 'controllers/ProductsController.php';
$ct = new ProductsController($app);
print $ct->editAction($_GET['id']);
```

Tedy parametr "r" (jako route) se přeloží jako název třídy a název metody, která se zavolá s případnými dodatečnými parametry. K názvům z url se připojí "Controller" v případě třídy a "Action" v případě funkce.

Aplikaci tedy vytváříme psaním tříd jednotlivých controllerů a jejich akcí (a dalších případných pomocných tříd a knihoven). Doručení požadavků patřičné třídě obstará pclib.

(Ve skutečnosti $app->run() doručení provede pomocí třídy Router, která se dá upravit a tím definovat i složitější pravidla překladu url na volání nějaké akce a zpět)

Friendly url

Nyní si můžeme velmi lehce zapnout jednoduchá friendly-url - stačí do adresáře **demo-app** překopírovat soubor **.htaccess**, který najdete v instalačním balíčku v adresáři **pclib/install** (někdy je potřeba nastavit parametr RewriteBase uvnitř souboru .htaccess)

Tento soubor nedělá nic jiného, než že překládá url v podobě

demo-app/products/edit?id=100

na

demo-app/index.php?r=products/edit&id=100

Autoloader

Třída **system\Autoloader** nahrává objekty pclib bez nutnosti používat require (a dělá i některá další kouzla). Za normálních okolností funguje automaticky a nemusíte o ní vědět.

Můžete ji ale použít také k nahrávání vlastních objektů.

Vše je založeno na principu, že soubor je pojmenovaný stejně jako třída, kterou obsahuje.

Pak jednoduše dáme vědět o adresáři s našimi třídami autoloaderu:

```
$pclib->autoloader->addDirectory('libs/utils/');
$a = new MojeTrida; // hleda soubor libs/utils/MojeTrida.php
```

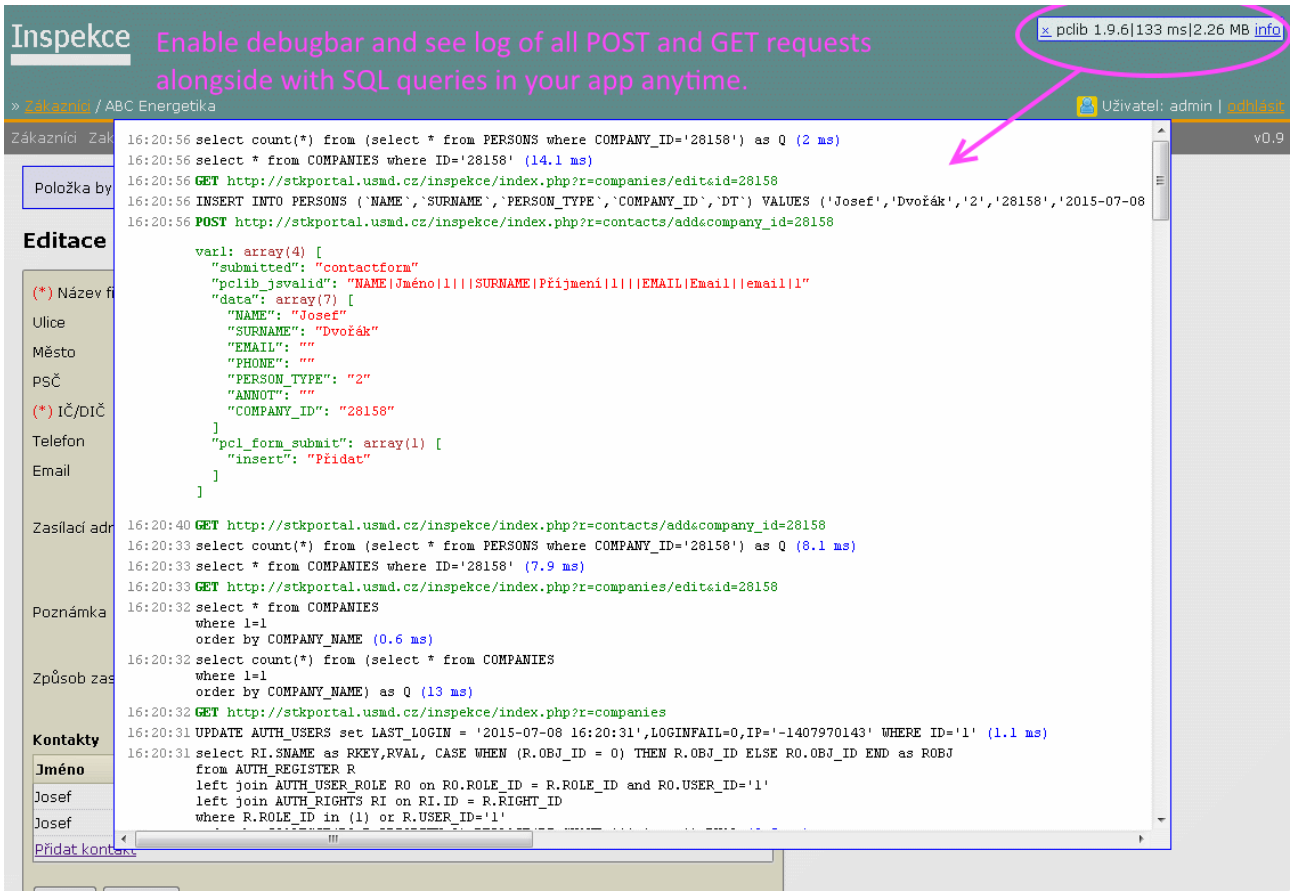
Debugger

Třída **Debugger** zobrazuje chybová hlášení a výpisy proměnných pomocí funkce **dump()**. Můžete ho konfigurovat pomocí odkazu $app->debugger. Příklad:

```
$app->debugger->maxLevel = 4;       //nastavi pocet vnoreni zobrazovany funkci
dump()
$app->debugger->showSource = true;  //zobrazit uryvek kodu kolem vyskytu chyby
$app->debugger->useHtml = false;    //vypnuti html zobrazeni, napr. kvuli
logovani do souboru
```

Používáte-li kontrollery tj. **$app->run()** můžete využít také debugovací lištu.
Zapne se příkazem:

```
$app->debugMode = true;
```



Obsahuje historii všech dotazů odeslaných databázi a všech odeslaných požadavků GET, POST nebo AJAX. Při odeslání formuláře zaznamená i odeslaná pole. Jako úložiště využívá $app->logger.

(Je nutné mít v aplikaci volání $app->run() protože debugovací lišta získává data pomocí speciálních ajax požadavků jako pclib/debuglog, které se vyhodnotí právě ve funkci run())

Controllers - zastavení druhé

Kontroler je třída odvozená od třídy pclib\Controller. Ukázka:

```
//--- soubor controllers/ProductsController.php ---

class ProductsController extends BaseController
{
	function indexAction() {
		return "<h1>Seznam produktů</h1>";
	}

	function addAction() {
		return new PCForm('tpl/products/form.tpl');
	}
}
```

Zadáním **?r=products/add** se zavolá funkce **addAction()** a vrácená hodnota se vloží do šablony layoutu aplikace. Příkaz app->out() celý výstup vypíše.

Funkce musí mít postfix **Action** z toho důvodu, aby nebylo možné zavolat z prohlížeče jiné pomocné funkce, které v controlleru mohou být, ale nemají být přímo přístupné uživateli.

Vrácená hodnota může být string, nebo objekt, který se umí vypsat ve stringovém kontextu, jako třeba Form nebo Grid.

indexAction() a defaultAction()

Pokud zadáme **?r=products** použije se pro výpis akce pojmenovaná **indexAction()**

Pokud zadáme neexistující akci **?r=products/pokus** dojde k chybě:

-> Page not found: products/pokus

Současně se odešle http hlavička **404 Not Found**.

Můžete si ale napsat obsluhu neexistujících akcí sami definováním funkce defaultAction(). Pokud existuje, zavolá se v případě neexistující akce ona:

```
function defaultAction() {
	$this->app->error('Neexistující akce '. $this->app->action);
}
```

Ve třídě controlleru se můžete všude odkazovat na objekt $app pomocí atributu **$this->app**.

BaseController

Možná že jste si všimli, že náš objekt je odvozený od třídy BaseController. Je užitečnou technikou nedědit controller přímo od třídy pclib\Controller, ale udělat si třídu BaseController jako předka všech kontrolerů, byť by byla zpočátku prázdná.

V této třídě si můžete vytvořit funkce nebo atributy, které budou společné pro všechny controllery.

```
class BaseController extends pclib\Controller
{
	protected $db;
	protected $loggedUser;

	function init() {
		$this->db = $this->app->db;
		$this->loggedUser = $this->app->auth->loggedUser;
	}
}
```

Funkce **init()** se volá vždy bezprostředně po vytvoření controlleru třídou App. Můžete i změnit vlastnosti základní třídy Controller předefinováním jejích funkcí. Např. **$app->run()** volá ve skutečnosti metodu **$controller->run($action)**, která teprve rozhodne, co se bude dít.

```
function run($action) {
	if ($action->method == 'special') { ... }
	return parent::run($action);
}
```

Akci kontroleru lze také zavolat v libovolné šabloně:

```
<?elements
action PRODUCT_LIST route "products"
?>
... {PRODUCT_LIST} ...
```

(Vloží do šablony výstup funkce indexAction(). Pro akci showAction($id) bychom napsali ... route "products/show/id:{ID}")

Parametry akcí

Pclib inteligentně zanalyzují parametry vaší akce a přiřadí jim odpovídající hodnoty z požadavku.

```
class ArticlesController
{
	function showAction($name) {
		return 'Zobrazuji článek '.$name;
	}
}
```

?r=articles/show

-> Required parameter "name" for page "Articles/show" missing

?r=articles/show?name=novinky-v-regionu

-> Zobrazuji článek novinky-v-regionu

Samozřejmě je možné mít parametrů i několik, popřípadě některým nastavit default hodnoty, čímž se stanou nepovinné.

Zabezpečení

Globální nastavení zabezpečení řídí v pclib následující parametr:

'pclib.security' => array('tpl-escape' => false, 'csrf' => false, 'form-prevent-mass' => false),

V základním nastavení jsou tyto funkce vypnuté, částně pro zachování zpětné kompatibility, částečně proto, že použití vyžaduje dodatečnou režii.

SQL-injection

Implicitně pclib provádí ošetření dat vkládaných do SQL dotazů (ochrana proti sql-injection). Proto používejte důsledně pclib parametry {PARAMETR}, které se vždy ošetřují na výskyt nebezpečných znaků.

Na rozdíl od jiných frameworků pclib neobaluje automaticky parametr uvozovkami a proto ho **musíte použít vždy v uvozovkách**, jinak je ochrana nefunkční. Správně tedy:

```
$db->select("select * from ARTICLES where NAME='{0}'", $name);
```

XSS: Cross site scripting

Jedná se o vložení javascript kódu, který provede škodlivou operaci při svém zobrazení. Obrana spočívá v ošetření každého textu, který mohl zadat někdo zvenčí, při zobrazení šablony.

Problém může nastat například při provozu nějakého veřejného blogu, kam může útočník vložit javascript odesláním komentáře.

Lze ošetřit pole šablony individuálně:

```
string COMMENT escape
```

ale protože na to může programátor zapomenout, bývá doporučováno zapnout ošetření globálně. Zapnutí se provede nastavením parametru **tpl-escape => true**

To způsobí, že každé pole šablony bude automaticky ošetřeno. Pokud chcete udělat výjjimku, můžete použít atribut noescape:

```
string CONTENT noescape
```

Ošetření se provádí pomocí funkce **Tpl->escapeHtmlFunction**, která zavolá php funkci htmlspecialchars(). Ta escapuje html, takže se všechny tagy vypíšou jako obyčejný text.

Je možné napsat funkci, která bezpečné html jako **Nadpis** propustí a nebezpečné odfiltruje, ale je to poměrně složité. V PHP k tomu slouží například knihovna HTML Purifier.

Nastavení vlastní funkce na vyčištění html:

```
function escapeHtml($s) { return strip_tags($s); }
$template->escapeHtmlFunction = 'escapeHtml';
```

CSRF

Jedná se o útok, kdy v případě, že jste přihlášení do aplikace a navštívíte stránku útočníka, nebo se jinak spustí jeho kód, útočník může vygenerovat požadavek, který se provede s vašimi oprávněními.

Obrana spočívá v provádění citlivých operací pouze pomocí formuláře (ne GETem) se zapnutou csrf ochranou. Veřejně přístupné formuláře není třeba chránit.

Ochrana přidá do formuláře skryté pole **csrf_token** s kódem, který se ověří proti kódu uloženému v sessions, které je nutné mít zapnuté.

Lze chránit jednotlivé formuláře přidáním parametru do elements:

```
<?elements
class form ... csrf
...
```

Parametr **pclib.security csrf** zapne ochranu pro všechny formuláře.

Mass assignment vulnerability

Při odeslání formuláře může útočník podvrhnout pole, která ve formuláři nejsou, a ty se uloží do tabulky. Například pokud máme registrační formulář s poli JMENO, HESLO a v tabulce jsou sloupce JMENO, HESLO, ADMINISTRATOR default "0", může útočník nastavit pole administrátor a získat tak neoprávněně admin účet.

S parametrem **form-prevent-mass** se uloží pouze pole, která jsou výslovně uvedena v elements šablony.

Obecné zásady

- Validovat a "sanitizovat" všechny vstupy od uživatele. Kód:

  ```
  //velmi nebezpecne
  $picture = $_GET['picture'];
  exec("convert -thumbnail 80x80 images/$picture images/thumbs/$picture");
  ```

  je prosba o hacknutí celého serveru.

- Ujistěte se, že uživatel nemůže přistupovat k dokumentům jiného uživatele např. změnou id v url:

  ```
  faktury/show?id=123
  ```

- Přidejte do adresáře **tpl/** (nebo i jinam) soubor **.htaccess** s kódem:

  ```
  order allow,deny
  deny from all
  ```

  To zabrání zobrazit šablonu (jakýkoliv soubor v adresáři) zadáním úplné cesty s názvem souboru v prohlížeči.

- Pclib přidávají k názvům souborů, nahraných pomocí třídy **Form,** osmimístný hash, který zabraňuje zobrazení souboru kýmkoliv zadáním cesty k němu v prohlížeči.

  Soubor by neměl být zobrazován odkazem na něj, ale prostřednictvím php:

  ```
  filedata('data/uploaded/faktura-xXUu0Ja8.pdf', false, 'faktura.pdf');
  ```

Pár novinek: Form a Grid

V šabloně formuláře lze použít parametr **table** s názvem db-tabulky, který automaticky nastaví maxlength polí formuláře tak, aby nedošlo k přetečení velikosti pole v databázi.

```
<?elements
class form ... table "PRODUCTS"
...
```

Data zobrazená v gridu lze vyexportovat ve formátu csv:

```
function exportAction() {
      $grid = new PCGrid('tpl/products/grid.tpl', 'products');
      $grid->setQuery('select * from PRODUCTS');
      $grid->exportCsv('products.csv');
}
```

TemplateFactory

Třída **TemplateFactory** umí vytvořit form nebo grid z databázové tabulky. Pclib ji interně používají např. při tvorbě "rychlogridu" - potřebujete rychle datagrid třeba pro vytvoření hrubého návrhu a nemáte čas zrovna vyrábět šablonu.

```
//Vyrobí grid s defaultní šablonou

$grid = new PCGrid;
$grid->setQuery('select * from PRODUCTS');
print $grid;
```

Podobně lze zobrazit i form. Jak bude vypadat výstup určuje seznam sloupců z databáze a šablona pclib/assets/default-grid.tpl.

Třeba se vám někdy bude hodit použít TemplateFactory samostatně:

```
use pclib\extensions\TemplateFactory;

$columns = $db->columns('PRODUCTS');
$form = TemplateFactory::create('tpl/form-generator.tpl', $columns);
print $form;
```

Další verze

V pclib 2.2 se chystá především možnost nahrávat soubory pomocí třídy FileStorage, podporující například nový html5 parametr **multiple** pro upload více souborů.

Současný přístup je nevhodný, pokud vaše aplikace obsahuje document management s velkým množstvím souborů. FileStorage nabízí mnohem flexibilnější přístup.

```
$app->fileStorage = new pclib\FileStorage('data/uploaded');
```

(Všechny soubory se budou ukládat do adresáře data/uploaded, do podadresářů ve tvaru rok/měsíc. Tedy například: data/uploaded/2016/12/faktury_Cqzj4R8i.pdf)

Pravděpodobně bude aktualizována i validace a umožní např. přidávat individuální chybové hlášky k polím, nebo vytvářet vlastní validační pravidla.

Poznámka: Nemělo by dojít k porušení zpětné kompatibility - původní systém uploadu zůstává zachován pokud nezapnete nový, validační pravidla by měla zachovat všechna stávající + přidat nová.