

---

# Phalcon Framework Documentation

*Выпуск 1.3.0*

Phalcon Team

18 May 2014



<b>1</b>	<b>Что такое Phalcon?</b>	<b>3</b>
<b>2</b>	<b>Оглавление</b>	<b>5</b>
2.1	Наша мотивация . . . . .	5
2.2	Тест производительности фреймворков . . . . .	7
2.3	Установка . . . . .	22
2.4	Урок 1: Рассмотрим на примере . . . . .	41
2.5	Tutorial 2: Explaining INVO . . . . .	52
2.6	Урок 3: Создание простейшего REST API . . . . .	69
2.7	Список примеров . . . . .	77
2.8	Использование Dependency Injection . . . . .	77
2.9	Архитектура MVC . . . . .	94
2.10	Использование контроллеров . . . . .	95
2.11	Работа с Моделями . . . . .	102
2.12	Язык запросов Phalcon (PHQL) . . . . .	158
2.13	Кэширование в ORM . . . . .	174
2.14	ODM (Object-Document Mapper) . . . . .	189
2.15	Использование представлений (Views) . . . . .	200
2.16	Помощники представлений . . . . .	218
2.17	Управление ресурсами (Assets Management) . . . . .	228
2.18	Шаблонизатор Volt . . . . .	234
2.19	MVC Приложения . . . . .	256
2.20	Маршрутизация (Routing, Роутинг) . . . . .	265
2.21	Диспетчер контроллеров . . . . .	281
2.22	Микроприложения . . . . .	285
2.23	Работа с пространством имён . . . . .	295
2.24	Менеджер событий EventsManager . . . . .	297
2.25	Заголовки запроса (Request) . . . . .	304
2.26	Заголовки ответа (Responses) . . . . .	307
2.27	Управление Куками . . . . .	310
2.28	Генерация ссылок (URLs) . . . . .	311
2.29	Информационные сообщения . . . . .	314
2.30	Сохранение данных в сессии . . . . .	317
2.31	Фильтрация и очистка . . . . .	320
2.32	Контекстное экранирование . . . . .	324
2.33	Валидация . . . . .	327
2.34	Формы . . . . .	332
2.35	Чтение конфигурации . . . . .	341

2.36	Постраничная навигация (Paginators) . . . . .	343
2.37	Улучшение производительности с помощью Кэширования . . . . .	346
2.38	Безопасность . . . . .	353
2.39	Зашифрование и расшифрование . . . . .	356
2.40	Списки Контроля Доступа (ACL) . . . . .	358
2.41	Поддержка многоязычности . . . . .	362
2.42	Универсальный загрузчик классов . . . . .	364
2.43	Логирование . . . . .	368
2.44	Парсер аннотаций . . . . .	372
2.45	Консольные приложения . . . . .	377
2.46	Очереди . . . . .	381
2.47	Уровень абстракции баз данных . . . . .	382
2.48	Интернационализация . . . . .	395
2.49	Миграции базы данных . . . . .	397
2.50	Отладка приложений . . . . .	404
2.51	Инструменты разработчика Phalcon (Developer Tools) . . . . .	408
2.52	Повышение производительности: Что дальше? . . . . .	431
2.53	Модульное тестирование (Unit test) . . . . .	438
2.54	API Indice . . . . .	442
2.55	Лицензия . . . . .	894
<b>3</b>	<b>Документация в других форматах</b>	<b>895</b>

Phalcon приветствует вас, мы смотрим по новому на построение фреймворков. Наша миссия заключается в создании продвинутого инструмента для разработки веб-сайтов и приложений, не беспокоясь о производительности.



### Что такое Phalcon?

---

Phalcon это проект с открытым исходным кодом, полноценный фреймворк для PHP 5 в виде Си-расширения, оптимизированного для высокой производительности. Для работы вам не нужно знать язык Си, так как функциональность фреймворка представлена классами PHP и полностью готова к использованию. Части Phalcon очень слабо связаны между собой, что позволяет использовать их в качестве независимых компонентов для любого вашего приложения.

Phalcon заботится не только о производительности, нашей целью является создание надежного и простого в использовании инструмента!

Перевод документации на русский язык производится энтузиастами. Мы будем рады всем желающим поучаствовать в переводе и поиске ошибок. Перевод и координация осуществляется на [GitHub](#) проекта и в группе [Вконтакте](#).



## Оглавление

---

### 2.1 Наша мотивация

Сейчас существует множество PHP фреймворков, но Phalcon такой один (правда, поверьте).

Почти все программисты предпочитают использовать фреймворки. Это обусловлено тем, что они обеспечивают богатый функционал который уже готов для использования и поддерживают принцип DRY ( Don't repeat yourself - не повторяй себя). Тем не менее, фреймворки в большинстве своём состоят из множества файлов и сотен строк кода, которые должны подключаться и выполняться при каждом запросе пользователя. Это существенно замедляет работу приложения, и так же негативно влияет на скорость работы и эмоции конечного пользователя.

#### 2.1.1 Вопрос

Почему мы не можем получить фреймворк со всеми этими преимуществами, но лишенный недостатков или сводящий их к минимуму?

Так и появился Phalcon!

Последние несколько месяцев мы глубоко исследовали возможности PHP для любой оптимизации большой или маленькой. Поняв Zend Engine, мы смогли убрать лишние проверки, уменьшить код и выполнить такие низкоуровневые оптимизации, которые позволили добиться максимальной производительности от Phalcon.

#### 2.1.2 Почему?

- Фреймворки стали обязательными для профессионального развития и работы с PHP
- Фреймворки предлагают чёткую философию, лёгкую поддержку и написание кода, а саму работу делают увлекательной и приятной
- Мы любим PHP и думаем, что он может быть использован для создания более крупных и амбициозных проектов

#### 2.1.3 Как работает PHP внутри?

- PHP - язык динамический, в нём присутствует так называемый слабый контроль типов. Каждый раз, при выполнении бинарной операции (например,  $2 + "2"$ ), PHP сначала проверяет типы операндов для выполнения преобразований

- PHP каждый раз заново интерпретируется и не компилируется. В этом основная проблема и главная потеря производительности
- При каждом запросе все сценарии должны быть сначала интерпретированы
- Если не используются акселераторы, кэширующие PHP-код в скомпилированный байт-код (например APC), то при каждом запросе так же проверяется синтаксис каждого файла

#### 2.1.4 Как работают традиционные фреймворки на PHP?

- Много файлов с разными классами и функциями считаются при каждом обращении. Чтение файлов с диска пагубно влияет на производительность, особенно когда каталогов и файлов много
- Современные фреймворки используют так называемую ленивую или отложенную загрузку (авто-загрузку) для увеличения производительности (для сценария используются только необходимые файлы с кодом)
- Некоторые из этих файлов содержат классы и методы, которые не используются в каждом запросе, но они загружены всегда, потребляя при этом память
- Последовательная загрузка или интерпретация довольно сильно нагружает сервер и негативно влияет на производительность
- Код фреймворка меняется не часто, но приложение должно его каждый раз загружать, проверять и интерпретировать, каждый раз, на каждый запрос

#### 2.1.5 Как работает расширение PHP?

- Си расширение загружается один раз, вместе с загрузкой библиотеки или демона PHP
- Классы и функции расширения готовы к использованию всегда и в любых приложениях
- Код не интерпретируется, он уже скомпилирован и оптимизирован. При этом оптимизирован с учётом особенностей текущей системы и используемого процессора

#### 2.1.6 Как работает Phalcon?

- Компоненты фреймворка слабо связаны между собой. С Phalcon можно использовать всю базу, или только отдельные части
- Низкоуровневая оптимизация позволяет минимизировать накладные расходы на реализацию в приложении паттерна MVC
- Взаимодействие с базами данных реализовано на Си по технологии ORM и выдаёт максимальную производительность
- Phalcon напрямую обращается к внутренним структурам PHP, что позволяет делать все операции максимально быстро

#### 2.1.7 Почему вам нужен Phalcon?

Каждое приложение имеет свои требования и задачи. Некоторые, например, предназначены для выполнения ряда задач и создания контента, который редко меняется. Эти приложения могут быть созданы с помощью любого языка программирования или фреймворка. Использование кэша, как правило, делает такую реализацию быстрой, независимо от того, на сколько медленно или быстро работает приложение.

Другие приложения генерируют контент практически сразу, изменяя от запроса к запросу. В этом случае PHP используется для адресации всех запросов и генерирования содержимого. Это могут быть приложения с API-интерфейсом, дискуссионные форумы с высокой нагрузкой, блоги с большим количеством комментариев и участников, статистические приложения, панель администратора, планирование ресурсов предприятия (ERP), программное обеспечение для обработки данных в реальном времени и многое другое.

Приложение будет работать так медленно, на сколько медленно работает его самая медленная часть/процесс. Phalcon предлагает очень быструю, но функциональную базу, которая позволяет разработчикам сконцентрироваться на ускорении работы своих приложений/кода. В процессе соответствующей разработки Phalcon может обработать гораздо больше функций/запросов с меньшим потреблением памяти и циклов обработки.

### 2.1.8 Заключение

Phalcon это попытка сделать быстрейший фреймворк для PHP. Вы получаете возможность использовать очень простой и надёжный инструмент для создания быстрых приложений без проблем с производительностью. Наслаждайтесь!

## 2.2 Тест производительности фреймворков

Раньше производительность не была главным приоритетом при выборе фреймворка для веб приложений. Это могло компенсироваться соответствующим оборудованием. Однако, когда в Google [стали](#) учитывать скорость сайта при ранжировании поиска, производительность стала одним из приоритетов, наряду с функциональностью. Это еще один способ, который улучшение производительность приложения будет положительно влиять на интернет.

Контрольные показатели ниже, показывают, насколько эффективнее Phalcon по сравнению с другими традиционными PHP фреймворками. Эти критерии обновляются при выходе стабильных версий фреймворков или обновлении Phalcon.

Мы предлагаем всем программистам клонировать наши тесты, которые мы используем для проверок. Если у вас есть предложения по оптимизации или комментарии, пожалуйста, [напишите нам](#). Скачать исходники тестов на [Github](#)

### 2.2.1 Параметры тестовой среды

Для всех фреймворков использовался APC. Во избежание накладных расходов по возможности был отключен mod-rewrite в Apache.

Среда тестирования выглядит следующим образом:

- Операционная система: Mac OS X Lion 10.7.4
- Веб сервер: Apache httpd 2.2.22
- PHP: 5.3.15
- Процессор: 2.04 Ghz Intel Core i5
- Оперативная память: 4GB 1333 MHz DDR3
- Жесткий диск: 500GB SATA Disk

*Информация о PHP:*

*Настойки APC:*

**PHP Version 5.3.15**

<b>System</b>	Darwin Andress-MacBook-Pro.local 11.4.0 Darwin Kernel Version 11.4.0: Mon Apr 9 19:32:15 PDT 2012; root:xnu-1699.26.8~1/RELEASE_X86_64 x86_64
<b>Build Date</b>	Aug 15 2012 16:31:00
<b>Configure Command</b>	<code>./configure' '--prefix=/opt/local' '--mandir=/opt/local/share/man' '--infodir=/opt/local/share/info' '--program-suffix=53' '--includedir=/opt/local/include/php53' '--libdir=/opt/local/lib/php53' '--with-config-file-path=/opt/local/etc/php53' '--with-config-file-scan-dir=/opt/local/var/db/php53' '--disable-all' '--enable-bcmath' '--enable-ctype' '--enable-dom' '--enable-finfo' '--enable-filter' '--enable-hash' '--enable-json' '--enable-libxml' '--enable-pdo' '--enable-phar' '--enable-session' '--enable-simplexml' '--enable-tokenizer' '--enable-xml' '--enable-xmlreader' '--enable-xmlwriter' '--with-bz2=/opt/local' '--with-mhash=/opt/local' '--with-pcre-regex=/opt/local' '--with-libxml-dir=/opt/local' '--with-zlib=/opt/local' '--without-pear' '--disable-cgi' '--disable-cli' '--disable-fpm' '--with-apxs2=/opt/local/apache2/bin/apxs'</code>
<b>Server API</b>	Apache 2.0 Handler
<b>Virtual Directory Support</b>	disabled
<b>Configuration File (php.ini) Path</b>	/opt/local/etc/php53
<b>Loaded Configuration File</b>	/opt/local/etc/php53/php.ini
<b>Scan this dir for additional .ini files</b>	/opt/local/var/db/php53
<b>Additional .ini files parsed</b>	/opt/local/var/db/php53/APC.ini, /opt/local/var/db/php53/http.ini, /opt/local/var/db/php53/mysql.ini, /opt/local/var/db/php53/pear.ini
<b>PHP API</b>	20090626
<b>PHP Extension</b>	20090626
<b>Zend Extension</b>	220090626
<b>Zend Extension Build</b>	API220090626,NTS
<b>PHP Extension Build</b>	API20090626,NTS
<b>Debug Build</b>	no
<b>Thread Safety</b>	disabled
<b>Zend Memory Manager</b>	enabled
<b>Zend Multibyte</b>	disabled

**apc**

APC Support		enabled
<b>Version</b>	3.1.11	
<b>APC Debugging</b>	Disabled	
<b>MMAP Support</b>	Enabled	
<b>MMAP File Mask</b>	<i>no value</i>	
<b>Locking type</b>	spin Locks	
<b>Serialization Support</b>	php	
<b>Revision</b>	\$Revision: 325875 \$	
<b>Build Date</b>	Jul 20 2012 11:18:52	

Directive	Local Value	Master Value
apc.cache_by_default	On	On
apc.canonicalize	On	On
apc.coredump_unmap	Off	Off
apc.enable_cli	Off	Off
apc.enabled	On	On
apc.file_md5	Off	Off
apc.file_update_protection	2	2
apc.filters	<i>no value</i>	<i>no value</i>
apc.gc_ttl	3600	3600
apc.include_once_override	Off	Off
apc.lazy_classes	Off	Off
apc.lazy_functions	Off	Off
apc.max_file_size	1M	1M
apc.mmap_file_mask	<i>no value</i>	<i>no value</i>
apc.num_files_hint	1000	1000
apc.preload_path	<i>no value</i>	<i>no value</i>
apc.report_autofilter	Off	Off
apc.rfc1867	Off	Off
apc.rfc1867_freq	0	0
apc.rfc1867_name	APC_UPLOAD_PROGRESS	APC_UPLOAD_PROGRESS
apc.rfc1867_prefix	upload_	upload_
apc.rfc1867_ttl	3600	3600
apc.serializer	default	default
apc.shm_segments	1	1
apc.shm_size	32M	32M
apc.slam_defense	On	On
apc.stat	Off	Off
apc.stat_ctime	Off	Off
apc.ttl	0	0
apc.use_request_time	On	On

## 2.2.2 Список тестов

### Тест производительности Hello World

#### Цель тестирования

Мы создали тест “Hello World” для создания минимальной нагрузки на каждый фреймворк. Большинство не любит такие сравнения, потому что в реальных приложениях используются более сложные функции и структуры. Однако, данный тест позволяет выявить минимальное время, необходимое каждому фреймворку для выполнения одной простой задачи. Такая задача требует выполнения минимальных условий для работы каждого фреймворка.

По своей сути, тест только измеряет время, необходимое фреймворку для запуска, выполнения действия и освобождения ресурсов в конце работы. Любому PHP приложению с поддержкой архитектуры MVC на это требуется время. Выполняя такой простейший тест, мы можем быть уверены, что время, требуемое для более сложных операций, будет выше.

Для каждого фреймворка были созданы Контроллер и представление (view). Контроллер называется “say”, а выполняемое действие — “hello”. Контроллер только передает в представление данные для отображения строки (“Hello!”). Для тестирования была использована утилита “ab”, мы отправляли фреймворкам 2000 запросов с 10 одновременными подключениями.

#### Контрольные замеры

Параметры ниже были выбраны для сравнения производительности каждого фреймворка:

- Число обработанных запросов в секунду (Requests per second)
- Время на выполнение всех запросов в teste
- Число используемых файлов на один запрос (использована функция `get_included_files`).
- Использование памяти на запрос (использована функция `memory_get_usage`).

#### Соперники

- [Yii](#) (`YII_DEBUG=false`) (`yii-1.1.13`)
- [Symfony](#) (2.0.11)
- [Zend Framework](#) (1.11.11)
- [Kohana](#) (3.2.0)
- [FuelPHP](#) (1.2.1)
- [CakePHP](#) (2.1.3)
- [Laravel](#) 3.2.5
- [CodeIgniter](#) (2.1.0)

#### Результаты

##### Yii (`YII_DEBUG=false`) версии `yii-1.1.13`

```
# ab -n 2000 -c 10 http://localhost/bench/helloworld/yii/index.php?r=say/hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

```
Server Software:      Apache/2.2.22
Server Hostname:    localhost
Server Port:        80

Document Path:      /bench/helloworld/yii/index.php?r=say/hello
Document Length:   61 bytes

Concurrency Level:  10
Time taken for tests: 2.081 seconds
Complete requests:  2000
Failed requests:   0
Write errors:       0
Total transferred: 508000 bytes
HTML transferred:  122000 bytes
Requests per second: 961.28 [#/sec] (mean)
Time per request:   10.403 [ms] (mean)
Time per request:   1.040 [ms] (mean, across all concurrent requests)
Transfer rate:      238.44 [Kbytes/sec] received
```

Connection Times (ms)

	min	mean [+/-sd]	median	max	
Connect:	0	10	4.3	9	42
Processing:	0	0	1.0	0	24
Waiting:	0	0	0.8	0	17
Total:	3	10	4.3	9	42

Percentage of the requests served within a certain time (ms)

50%	9
66%	11
75%	13
80%	14
90%	15
95%	17
98%	21
99%	26
100%	42 (longest request)

## Symfony версии 2.1.6

```
# ab -n 2000 -c 10 http://localhost/bench/Symfony/web/app.php/say/hello/
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

```
Server Software:      Apache/2.2.22
Server Hostname:    localhost
Server Port:        80
```

```
Document Path:          /bench/Symfony/web/app.php/say/hello/
Document Length:        16 bytes

Concurrency Level:      5
Time taken for tests:   1.848 seconds
Complete requests:      1000
Failed requests:         0
Write errors:            0
Total transferred:      249000 bytes
HTML transferred:       16000 bytes
Requests per second:    541.01 [#/sec] (mean)
Time per request:       9.242 [ms] (mean)
Time per request:       1.848 [ms] (mean, across all concurrent requests)
Transfer rate:          131.55 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean [+/-sd]	median	max
Connect:	0	9	4.8	8
Processing:	0	0	0.6	0
Waiting:	0	0	0.6	0
Total:	4	9	4.8	61

#### Percentage of the requests served within a certain time (ms)

50%	8
66%	9
75%	11
80%	12
90%	15
95%	18
98%	22
99%	30
100%	61 (longest request)

### CodeIgniter версии 2.1.0

```
# ab -n 2000 -c 10 http://localhost/bench/codeigniter/index.php/say/hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

```
Server Software:        Apache/2.2.22
Server Hostname:        localhost
Server Port:            80

Document Path:          /bench/helloworld/codeigniter/index.php/say/hello
Document Length:        16 bytes

Concurrency Level:      10
Time taken for tests:   1.888 seconds
Complete requests:      2000
Failed requests:         0
Write errors:            0
Total transferred:      418000 bytes
HTML transferred:       32000 bytes
```

```
Requests per second:      1059.05 [#/sec] (mean)
Time per request:        9.442 [ms] (mean)
Time per request:        0.944 [ms] (mean, across all concurrent requests)
Transfer rate:           216.15 [Kbytes/sec] received
```

	min	mean	[+/-sd]	median	max
Connect:	0	9	4.1	9	33
Processing:	0	0	0.8	0	19
Waiting:	0	0	0.7	0	16
Total:	3	9	4.2	9	33

	Percentage of the requests served within a certain time (ms)
50%	9
66%	10
75%	11
80%	12
90%	14
95%	16
98%	21
99%	24
100%	33 (longest request)

## Kohana версии 3.2.0

```
# ab -n 2000 -c 10 http://localhost/bench/helloworld/kohana/index.php/say/hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

Server Software:	Apache/2.2.22
Server Hostname:	localhost
Server Port:	80
Document Path:	/bench/helloworld/kohana/index.php/say/hello
Document Length:	15 bytes
Concurrency Level:	10
Time taken for tests:	2.324 seconds
Complete requests:	2000
Failed requests:	0
Write errors:	0
Total transferred:	446446 bytes
HTML transferred:	30030 bytes
Requests per second:	860.59 [#/sec] (mean)
Time per request:	11.620 [ms] (mean)
Time per request:	1.162 [ms] (mean, across all concurrent requests)
Transfer rate:	187.60 [Kbytes/sec] received

	min	mean	[+/-sd]	median	max
Connect:	0	11	5.1	10	64
Processing:	0	0	1.9	0	39
Waiting:	0	0	1.4	0	35
Total:	3	11	5.3	11	64

Percentage of the requests served within a certain time (ms)

50%	11
66%	13
75%	15
80%	15
90%	17
95%	18
98%	24
99%	31
100%	64 (longest request)

## Fuel версии 1.2.1

```
# ab -n 2000 -c 10 http://localhost/bench/helloworld/fuel/public/say/Hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

```
Server Software:      Apache/2.2.22
Server Hostname:     localhost
Server Port:        80

Document Path:       /bench/helloworld/fuel/public/say/Hello
Document Length:    16 bytes

Concurrency Level:   10
Time taken for tests: 2.742 seconds
Complete requests:  2000
Failed requests:    0
Write errors:        0
Total transferred:  418000 bytes
HTML transferred:   32000 bytes
Requests per second: 729.42 [#/sec] (mean)
Time per request:   13.709 [ms] (mean)
Time per request:   1.371 [ms] (mean, across all concurrent requests)
Transfer rate:      148.88 [Kbytes/sec] received
```

Connection Times (ms)

	min	mean [+/-sd]	median	max
Connect:	0	13	6.0	12
Processing:	0	0	1.3	0
Waiting:	0	0	0.8	0
Total:	4	14	6.1	13

Percentage of the requests served within a certain time (ms)

50%	13
66%	15
75%	17
80%	17
90%	19
95%	24
98%	30
99%	38
100%	80 (longest request)

**Cake версии 2.1.3**

```
# ab -n 10 -c 5 http://localhost/bench/cake/say/hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient).....done

Server Software:      Apache/2.2.22
Server Hostname:      localhost
Server Port:          80

Document Path:        /bench/cake/say/hello
Document Length:      16 bytes

Concurrency Level:    5
Time taken for tests: 30.051 seconds
Complete requests:   10
Failed requests:      0
Write errors:          0
Total transferred:   1680 bytes
HTML transferred:    160 bytes
Requests per second:  0.33 [#/sec] (mean)
Time per request:    15025.635 [ms] (mean)
Time per request:    3005.127 [ms] (mean, across all concurrent requests)
Transfer rate:        0.05 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       0    2   3.6     0    11
Processing:  15009 15020   9.8  15019  15040
Waiting:       9   21   7.9    25    33
Total:        15009 15022   8.9  15021  15040

Percentage of the requests served within a certain time (ms)
 50% 15021
 66% 15024
 75% 15024
 80% 15032
 90% 15040
 95% 15040
 98% 15040
 99% 15040
100% 15040 (longest request)
```

**Zend Framework версии 1.11.11**

```
# ab -n 2000 -c 10 http://localhost/bench/helloworld/zendfw/public/index.php
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
```

```
Server Software:      Apache/2.2.22
Server Hostname:      localhost
```

```
Server Port: 80

Document Path: /bench/helloworld/zendfw/public/index.php
Document Length: 16 bytes

Concurrency Level: 10
Time taken for tests: 5.641 seconds
Complete requests: 2000
Failed requests: 0
Write errors: 0
Total transferred: 418000 bytes
HTML transferred: 32000 bytes
Requests per second: 354.55 [#/sec] (mean)
Time per request: 28.205 [ms] (mean)
Time per request: 2.820 [ms] (mean, across all concurrent requests)
Transfer rate: 72.36 [Kbytes/sec] received
```

```
Connection Times (ms)
      min   mean[+/-sd] median   max
Connect:      0    27   9.6     25    89
Processing:   0     1   3.0     0    70
Waiting:     0     0   2.9     0    70
Total:       9    28   9.6     26    90
```

```
Percentage of the requests served within a certain time (ms)
```

50%	26
66%	28
75%	32
80%	34
90%	41
95%	46
98%	55
99%	62
100%	90 (longest request)

## Laravel версии 3.2.5

```
# ab -n 2000 -c 10 http://localhost/bench/helloworld/laravel/public/say/hello
```

```
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

```
Benchmarking localhost (be patient)
```

```
Server Software: Apache/2.2.22
Server Hostname: localhost
Server Port: 80

Document Path: /bench/helloworld/laravel/public/say/hello
Document Length: 15 bytes

Concurrency Level: 10
Time taken for tests: 4.090 seconds
Complete requests: 2000
Failed requests: 0
Write errors: 0
```

```
Total transferred:      1665162 bytes
HTML transferred:      30045 bytes
Requests per second:   489.03 [#/sec] (mean)
Time per request:      20.449 [ms] (mean)
Time per request:      2.045 [ms] (mean, across all concurrent requests)
Transfer rate:         397.61 [Kbytes/sec] received
```

Connection Times (ms)				
	min	mean [+/-sd]	median	max
Connect:	0	20	7.6	19
Processing:	0	0	2.5	0
Waiting:	0	0	2.5	0
Total:	6	20	7.6	19
				93

Percentage of the requests served within a certain time (ms)	
50%	19
66%	21
75%	23
80%	24
90%	29
95%	34
98%	42
99%	48
100%	93 (longest request)

## Phalcon версии 0.8.0

```
# ab -n 2000 -c 10 http://localhost/bench/helloworld/phalcon/index.php?_url=/say/hello
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

```
Server Software:        Apache/2.2.22
Server Hostname:       localhost
Server Port:          80

Document Path:         /bench/helloworld/phalcon/index.php?_url=/say/hello
Document Length:       16 bytes

Concurrency Level:     10
Time taken for tests:  0.789 seconds
Complete requests:    2000
Failed requests:       0
Write errors:          0
Total transferred:    418000 bytes
HTML transferred:     32000 bytes
Requests per second:   2535.82 [#/sec] (mean)
Time per request:      3.943 [ms] (mean)
Time per request:      0.394 [ms] (mean, across all concurrent requests)
Transfer rate:         517.56 [Kbytes/sec] received
```

Connection Times (ms)				
	min	mean [+/-sd]	median	max
Connect:	0	4	1.7	3
Processing:	0	0	0.2	0
				6

```
Waiting:      0    0   0.2     0     6
Total:       2    4   1.7     3    23
```

Percentage of the requests served within a certain time (ms)

50%	3
66%	4
75%	4
80%	4
90%	5
95%	6
98%	8
99%	14
100%	23 (longest request)

**Графики** Первый график показывает, сколько запросов в секунду смог обработать каждый фреймворк. Второй график показывает среднее время выполнения всех запросов.

## Заключение

Уникальная структура Phalcon предоставляет исключительную производительность и превосходит все используемые в этом тесте фреймворки.

## Тест производительности микро фреймворков

### Цель тестирования

Мы создали тест “Hello World” для создания минимальной нагрузки на каждый фреймворк. Единообразный тест выполнялся с каждым фреймворком.

Используя маршрутизацию для HTTP метода “GET” мы передаем параметр в обработчик и возвращаем ответ “Hello \$name”.

### Контрольные замеры

Параметры ниже были выбраны для сравнения производительности каждого фреймворка:

- Число обработанных запросов в секунду (Requests per second)
- Время на выполнение всех запросов в тесте
- Число используемых файлов на один запрос (использована функция `get_included_files`).
- Использование памяти на запрос (использована функция `memory_get_usage`).

### Соперники

- Slim
- Silex

## Результаты

### Slim Framework

```
# ab -n 1000 -c 5 http://localhost/bench/micro/slim/say/hello/Sonny
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)

Server Software:      Apache/2.2.22
Server Hostname:      localhost
Server Port:          80

Document Path:        /bench/micro/slim/say/hello/Sonny
Document Length:      13 bytes

Concurrency Level:    5
Time taken for tests: 0.882 seconds
Complete requests:   1000
Failed requests:     0
Write errors:         0
Total transferred:   206000 bytes
HTML transferred:    13000 bytes
Requests per second: 1134.21 [#/sec] (mean)
Time per request:    4.408 [ms] (mean)
Time per request:    0.882 [ms] (mean, across all concurrent requests)
Transfer rate:       228.17 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:        0    4   2.4     4    33
Processing:     0    0   0.5     0    11
Waiting:        0    0   0.5     0    11
Total:          2    4   2.4     4    33

Percentage of the requests served within a certain time (ms)
  50%    4
  66%    4
  75%    5
  80%    5
  90%    6
  95%    8
  98%   12
  99%   14
100%   33 (longest request)
```

### Silex

```
# ab -n 1000 -c 5 http://localhost/bench/micro/silex/say/hello/Sonny
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/

Benchmarking localhost (be patient)
```

```
Server Software:      Apache/2.2.22
Server Hostname:     localhost
Server Port:         80

Document Path:       /bench/micro/silex/say/hello/Sonny
Document Length:    12 bytes

Concurrency Level:   5
Time taken for tests: 2.228 seconds
Complete requests:  1000
Failed requests:    0
Write errors:        0
Total transferred:  225000 bytes
HTML transferred:   12000 bytes
Requests per second: 448.75 [#/sec] (mean)
Time per request:   11.142 [ms] (mean)
Time per request:   2.228 [ms] (mean, across all concurrent requests)
Transfer rate:      98.60 [Kbytes/sec] received
```

#### Connection Times (ms)

	min	mean	[+/-sd]	median	max
Connect:	0	11	5.1	10	44
Processing:	0	0	1.1	0	26
Waiting:	0	0	1.1	0	26
Total:	5	11	5.1	10	45

#### Percentage of the requests served within a certain time (ms)

50%	10
66%	12
75%	13
80%	14
90%	17
95%	20
98%	25
99%	29
100%	45 (longest request)

## Phalcon 0.5.0

```
# ab -n 1000 -c 5 http://localhost/bench/micro/phalcon/say/hello/Sonny
This is ApacheBench, Version 2.3 <$Revision: 655654 $>
Copyright 1996 Adam Twiss, Zeus Technology Ltd, http://www.zeustech.net/
Licensed to The Apache Software Foundation, http://www.apache.org/
```

Benchmarking localhost (be patient)

```
Server Software:      Apache/2.2.22
Server Hostname:     localhost
Server Port:         80

Document Path:       /bench/micro/phalcon/say/hello/Sonny
Document Length:    12 bytes

Concurrency Level:   5
Time taken for tests: 0.397 seconds
Complete requests:  1000
```

```

Failed requests:          0
Write errors:            0
Total transferred:      205000 bytes
HTML transferred:       12000 bytes
Requests per second:    2516.74 [#/sec] (mean)
Time per request:       1.987 [ms] (mean)
Time per request:       0.397 [ms] (mean, across all concurrent requests)
Transfer rate:          503.84 [Kbytes/sec] received

Connection Times (ms)
                   min   mean[+/-sd] median   max
Connect:           0     2   0.9     2     11
Processing:        0     0   0.2     0      5
Waiting:           0     0   0.2     0      4
Total:             1     2   0.9     2     11

Percentage of the requests served within a certain time (ms)
 50%    2
 66%    2
 75%    2
 80%    2
 90%    3
 95%    4
 98%    5
 99%    5
100%   11 (longest request)

```

**Графики** Первый график показывает, сколько запросов в секунду смог обработать каждый фреймворк. Второй график показывает среднее время выполнения всех запросов.

### Заключение

Уникальная структура Phalcon предоставляет исключительную производительность и превосходит все используемые в этом тесте фреймворки.

### 2.2.3 Список изменений

Добавлено в версии 1.0: Обновление Mar-20-2012: Тесты переделаны с учетом apc.stat установленного в ВЫКЛ. Расширение информации

Изменено в версии 1.1: Обновление May-13-2012: Шаблонизация в Symfony была переделана с Twig на прямое использование PHP. Параметры конфигурации Yii были изменены в соответствии с рекомендациями.

Изменено в версии 1.2: Обновление May-20-2012: Для сравнения добавлен фреймворк Fuel.

Изменено в версии 1.3: Обновление Jun-4-2012: Для сравнения добавлен фреймворк Cake. Но это не отображено на графике, потому как требует 30 секунд для запуска 10 из 1000.

Изменено в версии 1.4: Обновление Ago-27-2012: PHP обновлён до 5.3.15, APC обновлён до 3.1.11, Yii обновлён до 1.1.12, Phalcon обновлён до 0.5.0, Добавлен Laravel, операционная система обновлена до Mac OS X Lion. Обновлено железо.

## 2.2.4 Внешние источники

- For Impatient Web Users, an Eye Blink Is Just Too Long to Wait
- Millionaires performance cases: Impact of performance
- How fast are we going now?
- Speed, performance and human perception

## 2.3 Установка

Расширения для PHP устанавливаются несколько иначе чем обычные библиотеки или php-фреймворки. Вы можете скачать готовый бинарный файл для своей системы, или собрать его их исходников самостоятельно.

Phalcon работает с PHP 5.3.1, но ошибки в старых версиях PHP вызывают утечки памяти, и для надёжной работы рекомендуем использовать как минимум PHP 5.3.11 или выше.

В PHP версиях ниже 5.3.9 есть ошибки, влияющие на безопасность, эти версии не рекомендуется использовать. [Подробнее](#)

### 2.3.1 Windows

Для использования Phalcon в среде Windows достаточно скачать DLL библиотеку и добавить в конце php.ini :

```
extension=php_phalcon.dll
```

Перезапустить веб-сервер.

Существует обучающий скринкаст с пошаговой установкой Phalcon на Windows:

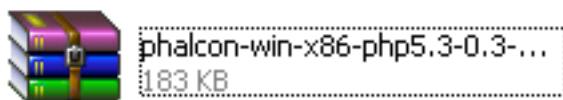
#### Краткое руководство

#### Установка на XAMPP

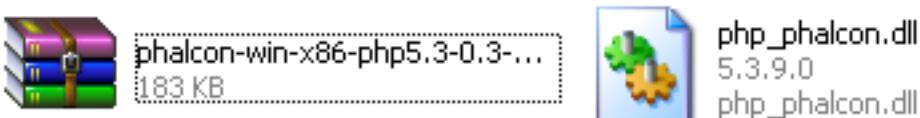
XAMPP представляет собой лёгкий вариант установки Apache в комплекте с MySQL, PHP и Perl. Просто скачав XAMPP его сразу можно использовать. Ниже представлена детальная инструкция по установке Phalcon на XAMPP для Windows. Крайне рекомендуем использовать последние версии XAMPP.

**Скачайте правильную версию Phalcon** XAMPP всегда выпускается с 32 разрядными версиями Apache и PHP. Вам необходимо так же скачивать x86 версию Phalcon для Windows в разделе скачиваний.

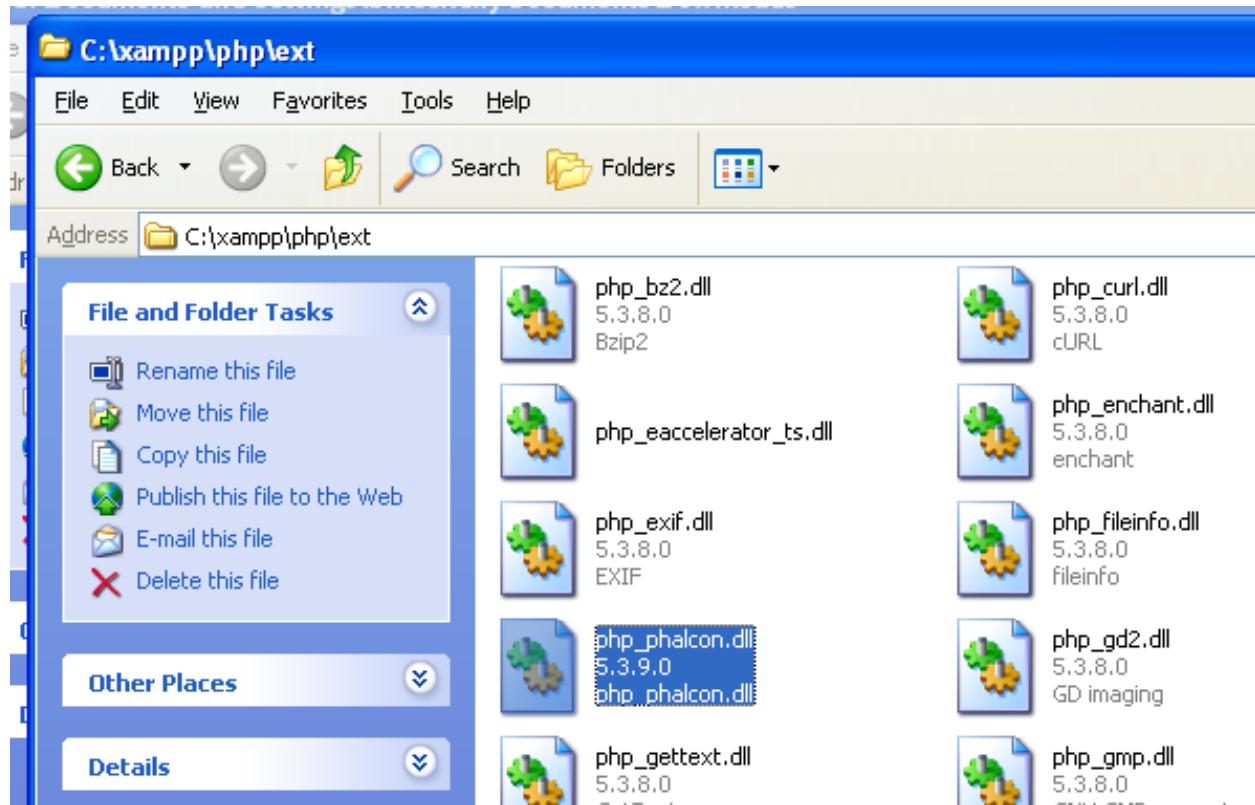
После скачивания библиотеки Phalcon у вас будет zip файл, примерно такой, как показано ниже:



Распакуйте архив и получите файл библиотеки Phalcon DLL:



Скопируйте файл php\_phalcon.dll в каталог PHP расширений. Если вы установили XAMPP в каталог c:\xampp, то расширения будут в c:\xampp\php\ext



Отредактируйте ваш файл php.ini, он располагается в C:\xampp\php\php.ini. Для редактирования можно использовать Блокнот или любую подобную программу. Мы рекомендуем использовать Notepad++ для избегания проблем с окончание и переводом строк. Добавьте в конец файла: extension=php\_phalcon.dll и сохраните его.

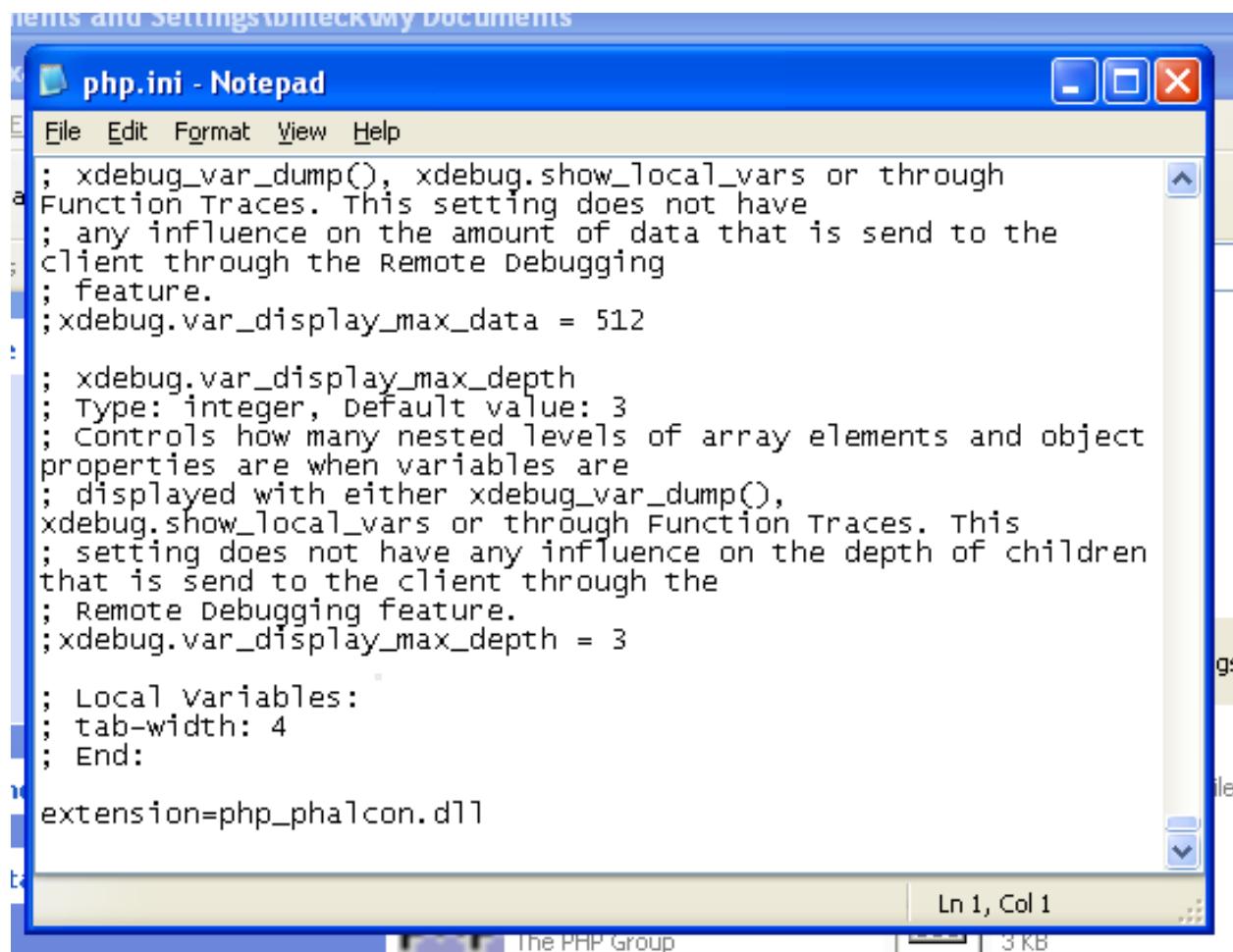
Перезапустите сервер Apache из контрольной панели XAMPP. PHP должен загрузиться с новой конфигурацией.

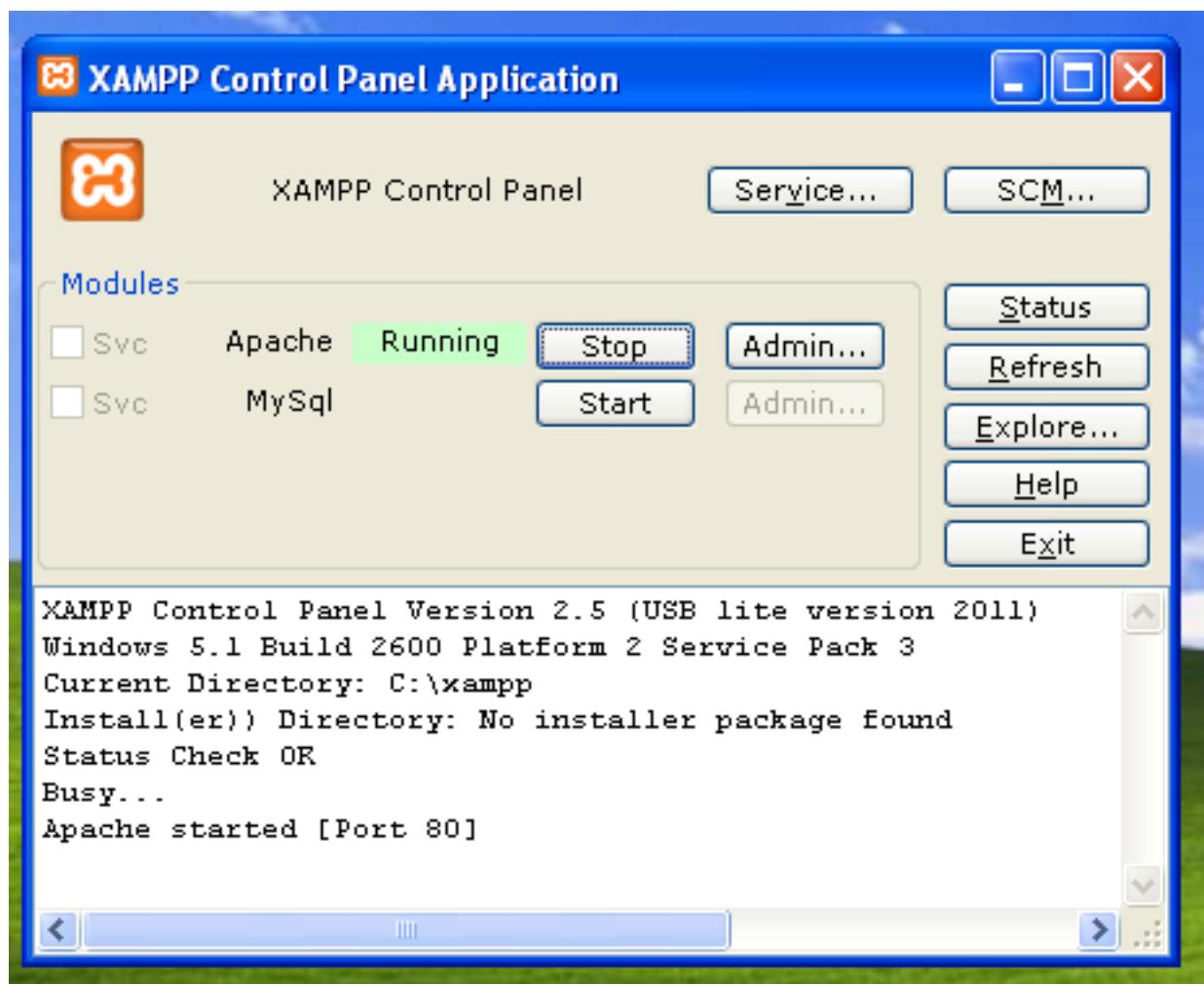
Откройте ваш браузер и перейдите на <http://localhost>. Должна появиться страница приветствия XAMPP. Нажмите на ссылку `phpinfo()`.

`phpinfo()` выводит обширную информацию о текущем состоянии PHP. Прокрутите страницу ниже и убедитесь что расширение phalcon загружено корректно.

Если вы увидели версию phalcon в выдаче `phpinfo()`, поздравляем!, вы готовы к полёту с Phalcon.

**Скринкаст** Нижеприведённый скринкаст отображает пошаговую установку Phalcon на Windows:







**B-LITE**  
1.7.7  
[PHP: 5.3.8]

Welcome  
Status  
Security  
Documentation  
Components

**PHP**  
phpinfo()  
CD Collection

**phalcon**

PDO support	enabled
PDO drivers	no value

**Version** 0.3.1

**Phar**

Phar: PHP Archive support	enabled
Phar EXT version	2.0.1
Phar API version	1.1.1

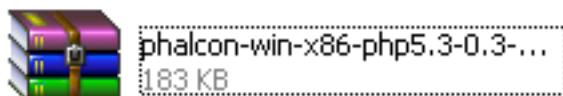
## Дополнительные руководства

- *Информация по установке*
- *Подробная установка на WAMP для Windows*

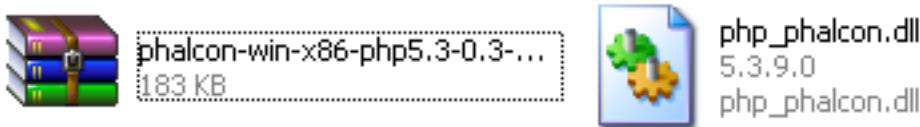
### Установка на WAMP

WampServer это платформа для веб-разработки под Windows. Он позволяет создавать веб-приложения с использованием Apache2, PHP и MySQL. Ниже представлена детальная инструкция по установке Phalcon на WampServer для Windows. Крайне рекомендуем использовать последние версии WAMPP.

**Скачайте правильную версию Phalcon** WAMP существует в 32 и 64 битных версиях. В разделе скачивания выберите нужную версию Phalcon для Windows в зависимости от имеющейся архитектуры. После скачивания библиотеки Phalcon у вас будет zip файл, примерно такой как показано ниже:



Распакуйте архив и получите файл библиотеки Phalcon DLL:



Скопируйте файл php\_phalcon.dll в каталог PHP расширений. Если вы установили WAMP в каталог c:\wamp, то расширения будут в c:\wamp\bin\php\php5.3.10\ext

Отредактируйте ваш php.ini file, он располагается в C:\wamp\bin\php\php5.3.10\php.ini. Для редактирования можно использовать Блокнот или любую подобную программу. Мы рекомендуем использовать Notepad++ для избегания проблем с окончанием и переводом строк. Добавьте в конец файла: extension=php\_phalcon.dll и сохраните его.

Так же отредактируйте файл php.ini file, расположенный в C:\wamp\bin\apache\Apache2.2.21\bin\php.ini. Добавьте в самый конец файла: extension=php\_phalcon.dll и сохраните его.

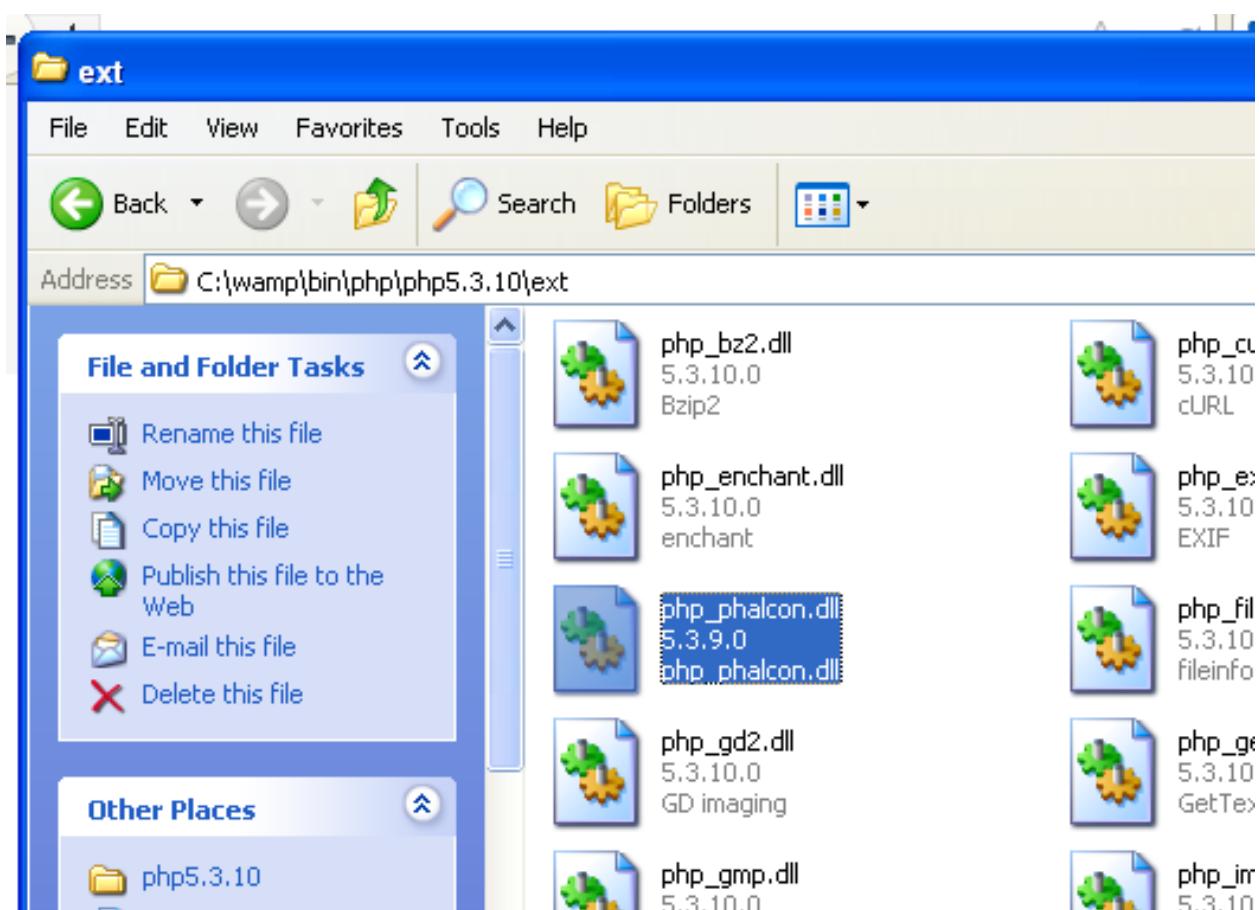
Перезапустите Apache. Кликните один раз на значок WampServer в системном трее. Выберите “Restart All Services” из выпадающего меню. Проверьте, что значок в трее снова стал зелёным.

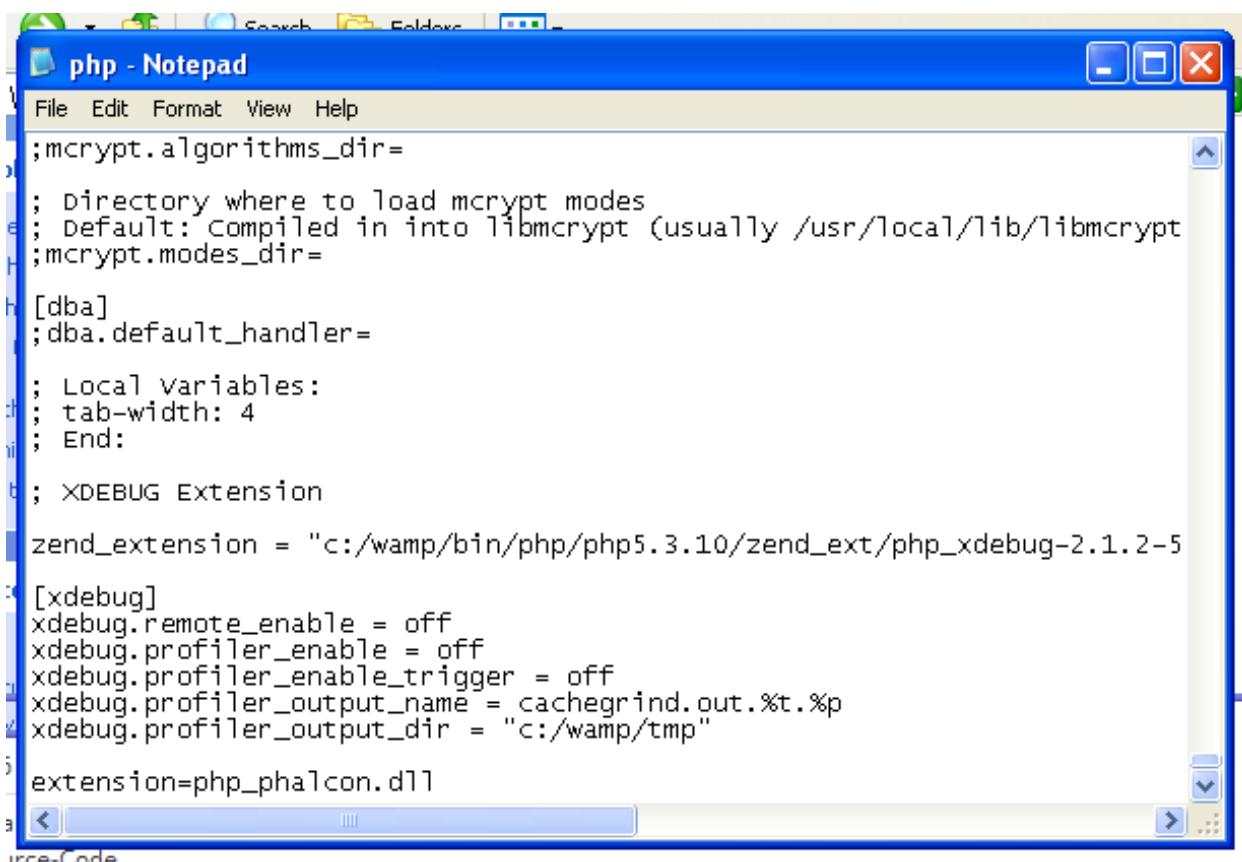
Откройте ваш браузер и перейдите на <http://localhost>. Должна появиться страница приветствия WAMPP. Найдите раздел “extensions loaded” и проверьте что расширение phalcon загружено.

Поздравляем!, вы готовы к полёту с Phalcon.

## Дополнительные руководства

- *Информация по установке*
- *Подробная установка на XAMPP для Windows*





```

File Edit Format View Help
;mcrypt.algorithms_dir=
; Directory where to load mcrypt modes
; Default: Compiled in into libmcrypt (usually /usr/local/lib/libmcrypt
;mcrypt.modes_dir=

[dba]
;dba.default_handler=

; Local Variables:
; tab-width: 4
; End:

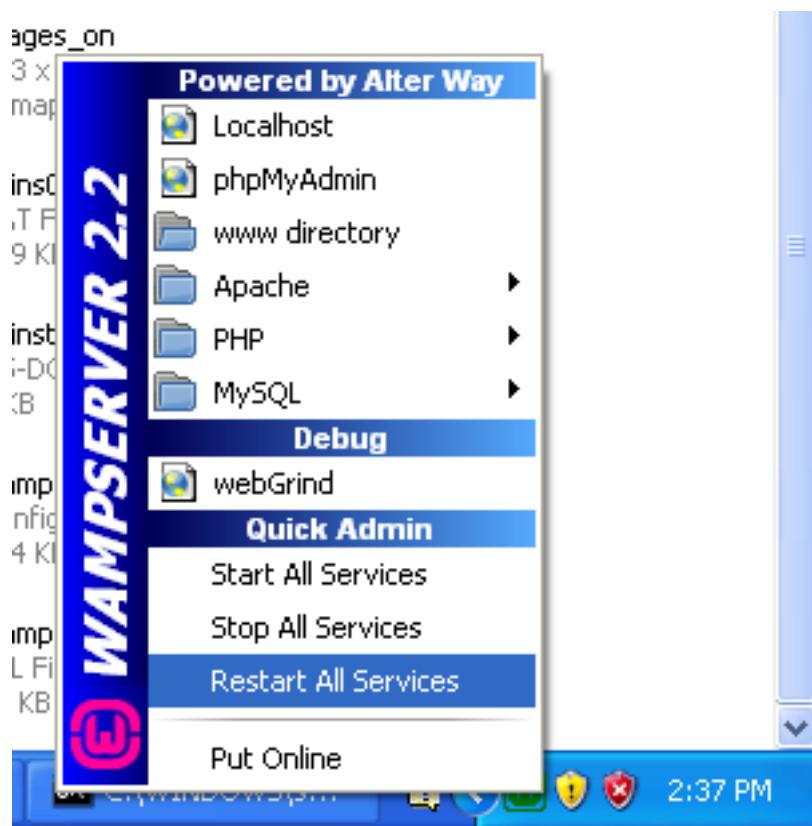
; XDEBUG Extension

zend_extension = "c:/wamp/bin/php/php5.3.10/zend_ext/php_xdebug-2.1.2-5

[xdebug]
xdebug.remote_enable = off
xdebug.profiler_enable = off
xdebug.profiler_enable_trigger = off
xdebug.profiler_output_name = cachegrind.out.%t.%p
xdebug.profiler_output_dir = "c:/wamp/tmp"

extension=php_phalcon.dll

```



## Server Configuration

**Apache Version :** 2.2.21

**PHP Version :** 5.3.10

<b>Loaded Extensions :</b>	<b>Core</b>	<b>bcmath</b>	<b>calendar</b>	<b>com_dotnet</b>
	<b>ctype</b>	<b>date</b>	<b>ereg</b>	<b>filter</b>
	<b>ftp</b>	<b>hash</b>	<b>iconv</b>	<b>json</b>
	<b>mcrypt</b>	<b>SPL</b>	<b>odbc</b>	<b>pcre</b>
	<b>Reflection</b>	<b>session</b>	<b>standard</b>	<b>mysqlnd</b>
	<b>tokenizer</b>	<b>zip</b>	<b>zlib</b>	<b>libxml</b>
	<b>dom</b>	<b>PDO</b>	<b>Phar</b>	<b>SimpleXML</b>
	<b>wddx</b>	<b>xml</b>	<b>xmlreader</b>	<b>xmlwriter</b>
	<b>apache2handler</b>	<b>embstring</b>	<b>gd</b>	<b>mysql</b>
	<b>mysqli</b>	<b>pdo_mysql</b>	<b>pdo_sqlite</b>	<b>phalcon</b>
	<b>mhash</b>	<b>xdebug</b>		

**MySQL Version :** 5.5.20

### 2.3.2 Linux/Solaris/Mac

Пользователи Linux/Solaris/Mac могут просто собрать Phalcon из исходных файлов:

#### Требования

Необходимы пакеты:

- Пакеты для разработки PHP 5.3.x/5.4.x/5.5.x
- Компилятор GCC (Linux/Solaris) или Xcode (Mac)
- Git (если он не установлен - пакет можно загрузить с GitHub и закачать на свой сервер по FTP/SFTP)

Специфичные пакеты для разных платформ:

```
#Ubuntu
sudo apt-get install git-core gcc autoconf
sudo apt-get install php5-dev php5-mysql
```

```
#Suse
sudo zypper -i gcc make autoconf2.13
sudo zypper -i php5-devel php5-pear php5-mysql
```

```
#CentOS/RedHat
sudo yum install gcc make
sudo yum install php-devel
```

```
#Solaris
pkg install gcc-45
pkg install php-53 apache-php53
```

## Компиляция

Создание расширения:

```
git clone git://github.com/phalcon/cphalcon.git
cd cphalcon/build
sudo ./install
```

Добавьте его в php.ini

```
extension=phalcon.so
```

Перезапустите веб-сервер.

При компиляции Phalcon сам выявляет тип платформы, но можно указать и явно:

```
sudo ./install 32bits
sudo ./install 64bits
sudo ./install safe
```

### 2.3.3 FreeBSD

Порт доступен для FreeBSD. Для установки служат простые команды:

```
pkg_add -r phalcon
```

или

```
export CFLAGS="-O2 -fno-delete-null-pointer-checks"
cd /usr/ports/www/phalcon && make install clean
```

### 2.3.4 Замечания по установке

Установка на разные веб-сервера:

#### Установка на Apache

Apache - популярный веб-сервер, доступный на большинстве современных платформ.

##### Конфигурация Apache для Phalcon

Следующие инструкции позволяют настроить Phalcon на Apache. В основном они сводятся к настройке поведения модуля mod-rewrite позволяющего использовать человеко-понятные URL ('ЧПУ') и *роутера компонентов*. Типичное приложение имеет следующую структуру:

```
test/
  app/
    controllers/
    models/
    views/
  public/
    css/
    img/
    js/
    index.php
```

**Корневой каталог документов** Самый распространённый случай - когда приложение устанавливается в любой подкаталог корневой директории. В таких случаях мы используем два .htaccess файла. Первый будет скрывать код приложения и перенаправлять запросы к корню приложения (public/).

```
# test/.htaccess

<IfModule mod_rewrite.c>
  RewriteEngine on
  RewriteRule ^$ public/ [L]
  RewriteRule (.*) public/$1 [L]
</IfModule>
```

Второй .htaccess будет располагаться уже в каталоге public/, и будет перенаправлять все запросы на файл public/index.php:

```
# test/public/.htaccess

<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-d
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteRule ^(.*)$ index.php?url=/${1} [QSA,L]
</IfModule>
```

Если нет желания или возможности использовать файлы .htaccess, то параметры также можно прописать в главном файле конфигурации Apache:

```
<IfModule mod_rewrite.c>

  <Directory "/var/www/test">
    RewriteEngine on
    RewriteRule ^$ public/ [L]
    RewriteRule (.*) public/$1 [L]
  </Directory>

  <Directory "/var/www/test/public">
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php?url=/${1} [QSA,L]
  </Directory>

</IfModule>
```

**Виртуальные хосты** Параметры можно также прописать в настройках конкретного виртуального хоста:

```
<VirtualHost *:80>

    ServerAdmin admin@example.host
    DocumentRoot "/var/vhosts/test/public"
    DirectoryIndex index.php
    ServerName example.host
    ServerAlias www.example.host

    <Directory "/var/vhosts/test/public">
        Options All
        AllowOverride All
        Allow from all
    </Directory>

</VirtualHost>
```

## Установка на Nginx

Nginx это свободный, с открытым исходным кодом, высокопроизводительный HTTP-сервер и прокси-сервер, а также IMAP/POP3 прокси-сервер. В отличие от традиционных серверов, Nginx не использует потоки для обработки запросов. Вместо этого он использует гораздо более масштабируемую управляемую событиями (асинхронную) архитектуру. Эта архитектура под высокой нагрузкой использует небольшой, и главное, предсказуемый объем памяти.

“PHP-FPM” (Менеджер процессов FastCGI) обычно используется для обработки PHP-файлов в Nginx. В настоящее время, PHP-FPM идет в комплекте с любым дистрибутивом PHP в Unix. Связка Phalcon + Nginx + PHP-FPM предоставляет мощный набор инструментов, который позволяет добиться максимальной производительности ваших PHP-приложений.

### Конфигурация Nginx для Phalcon

Конфигурации ниже позволяют настроить Nginx для работы с Phalcon:

**Базовая конфигурация** Использование переменной `$_GET['url']` для URIs:

```
server {

    listen 80;
    server_name localhost.dev;

    index index.php index.html index.htm;
    set $root_path '/var/www/phalcon/public';
    root $root_path;

    try_files $uri $uri/ @rewrite;

    location @rewrite {
        rewrite ^/(.*)$ /index.php?_url=$1;
    }

    location ~ \.php {
        fastcgi_pass unix:/run/php-fpm/php-fpm.sock;
        fastcgi_index /index.php;

        include /etc/nginx/fastcgi_params;
    }
}
```

```
fastcgi_split_path_info      ^(.+\.php)(/.+)$;
fastcgi_param PATH_INFO      $fastcgi_path_info;
fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
}

location ~*/(css|img|js|flv|swf|download)/(.+)$ {
    root $root_path;
}

location ~ /\.ht {
    deny all;
}
}
```

Использование `$_SERVER['REQUEST_URI']` для URIs:

```
server {

    listen 80;
    server_name localhost.dev;

    index index.php index.html index.htm;
    set $root_path '/var/www/phalcon/public';
    root $root_path;

    location / {
        try_files $uri $uri/ /index.php;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.php)(/.+)$;
        fastcgi_pass 127.0.0.1:9000;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }

    location ~*/(css|img|js|flv|swf|download)/(.+)$ {
        root $root_path;
    }

    location ~ /\.ht {
        deny all;
    }
}
```

### Частный случай

```
server {
    listen      80;
    server_name localhost;

    charset     utf-8;

    #access_log  /var/log/nginx/host.access.log  main;
```

```

set $root_path '/srv/www/htdocs/phalcon-website/public';

location / {
    root $root_path;
    index index.php index.html index.htm;

    # if file exists return it right away
    if (-f $request_filename) {
        break;
    }

    # otherwise rewrite it
    if (!-e $request_filename) {
        rewrite ^(.+)\$ /index.php?_url=/\$1 last;
        break;
    }
}

location ~ \.php {
    # try_files \$uri =404;

    fastcgi_index /index.php;
    fastcgi_pass 127.0.0.1:9000;

    include fastcgi_params;
    fastcgi_split_path_info ^(.+\.\php)(/.+)\$;
    fastcgi_param PATH_INFO \$fastcgi_path_info;
    fastcgi_param PATH_TRANSLATED \$document_root\$fastcgi_path_info;
    fastcgi_param SCRIPT_FILENAME \$document_root\$fastcgi_script_name;
}

location ~* ^/(css|img|js|flv|swf|download)\$ {
    root $root_path;
}
}

```

**Конфигурация по хосту** Такая конфигурация позволит иметь разные конфигурации для разных хостов:

```

server {
    listen 80;
    server_name localhost;
    set $root_path '/var/www/\$host/public';
    root $root_path;

    access_log /var/log/nginx/\$host-access.log;
    error_log /var/log/nginx/\$host-error.log error;

    index index.php index.html index.htm;

    try_files \$uri \$uri/ @rewrite;

    location @rewrite {
        rewrite ^/(.*)\$ /index.php?_url=/\$1;
    }

    location ~ \.php {

```

```
# try_files      $uri =404;

fastcgi_index /index.php;
fastcgi_pass   127.0.0.1:9000;

include fastcgi_params;
fastcgi_split_path_info ^(.+\.\php)(/.+)$;
fastcgi_param PATH_INFO $fastcgi_path_info;
fastcgi_param PATH_TRANSLATED $document_root$fastcgi_path_info;
fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
}

location ~* ^/(css|img|js|flv|swf|download)/(.+)$ {
    root $root_path;
}

location ~ /\.ht {
    deny all;
}
}
```

## Установка на Cherokee

Cherokee это высокопроизводительный веб сервер. Он очень быстрый, гибкий и лёгкий в настройке.

### Конфигурация Cherokee для Phalcon

Cherokee имеет удобный графический интерфейс для настройки практически всех параметров, доступных в web-сервере. Чтобы запустить администрирование сервера, нужно выполнить команду ‘/path-to-cherokee/sbin/cherokee-admin’ с правами суперадмина (root).

Создайте новый виртуальный хост, для этого кликните на ‘vServers’, и добавьте виртуальный сервер: Добавленный виртуальный хост должен появиться на панели слева. На вкладке ‘Behaviors’ вы можете увидеть набор правил для данного сервера. Нажмите кнопку ‘Rule Management’. Снимите выбор (галочки) с ‘Directory /cherokee\_themes’ и ‘Directory /icons’:

С помощью мастера добавьте обработчик ‘PHP Language’. Это позволит запускать PHP приложения:

Обычно такое решение не требует дополнительной настройки. Добавьте еще одно правило (behavior), на этот раз в разделе ‘Manual Configuration’. В списке ‘Rule Type’ выберите ‘File Exists’, и убедитесь что опция ‘Match any file’ включена:

На вкладке ‘Handler’ выберите обработчик (handler) ‘List & Send’:

Отредактируйте правило ‘Default’ для включения возможностей URL-rewrite. Выберите ‘Redirection’, затем добавьте регулярное выражение ^(.\*)\$:

Убедитесь, что в “Behaviors” выставлен нужный порядок:

Запустите приложение в браузере:

## Установка на встроенный в PHP веб-сервер

В PHP версии 5.4.0 был добавлен встроенный веб-сервер, который можно использовать для разработки.

Для запуска сервера выполните команду:

Welcome to Cherokee Admin

Language: Choose

**Join the Cherokee Community!**

We believe Web infrastructure software should evolve towards efficiency, scalability, and management simplicity.

We also believe that Free and Open Source Software bring numerous benefits to users, developers, and society in general.

The Cherokee Project community heartily welcomes you to join it and help to achieve its goals.

[Learn more](#)

**Server Information**

Server is Running	<a href="#">Stop Server</a>
Hostname	phalcon.site
Config File	/opt/cherokee/etc/cherokee/cherokee.conf

**Remote Services**

<input checked="" type="checkbox"/> Enable Remote Services	<a href="#">Sign in</a>
--	-------------------------

**Processors**

3.06GHz, 1 Logical Processor	2%
------------------------------	----

**Memory**

1006MB	110.1MB Used, 896.6MB Free	10.9%
--------	----------------------------	-------

**Support**

- [Getting started](#)
- [Subscribe to mailing lists](#)
- [Report a bug](#)
- [Purchase commercial support](#)
- [About Cherokee](#)

**Proud Cherokee Users**

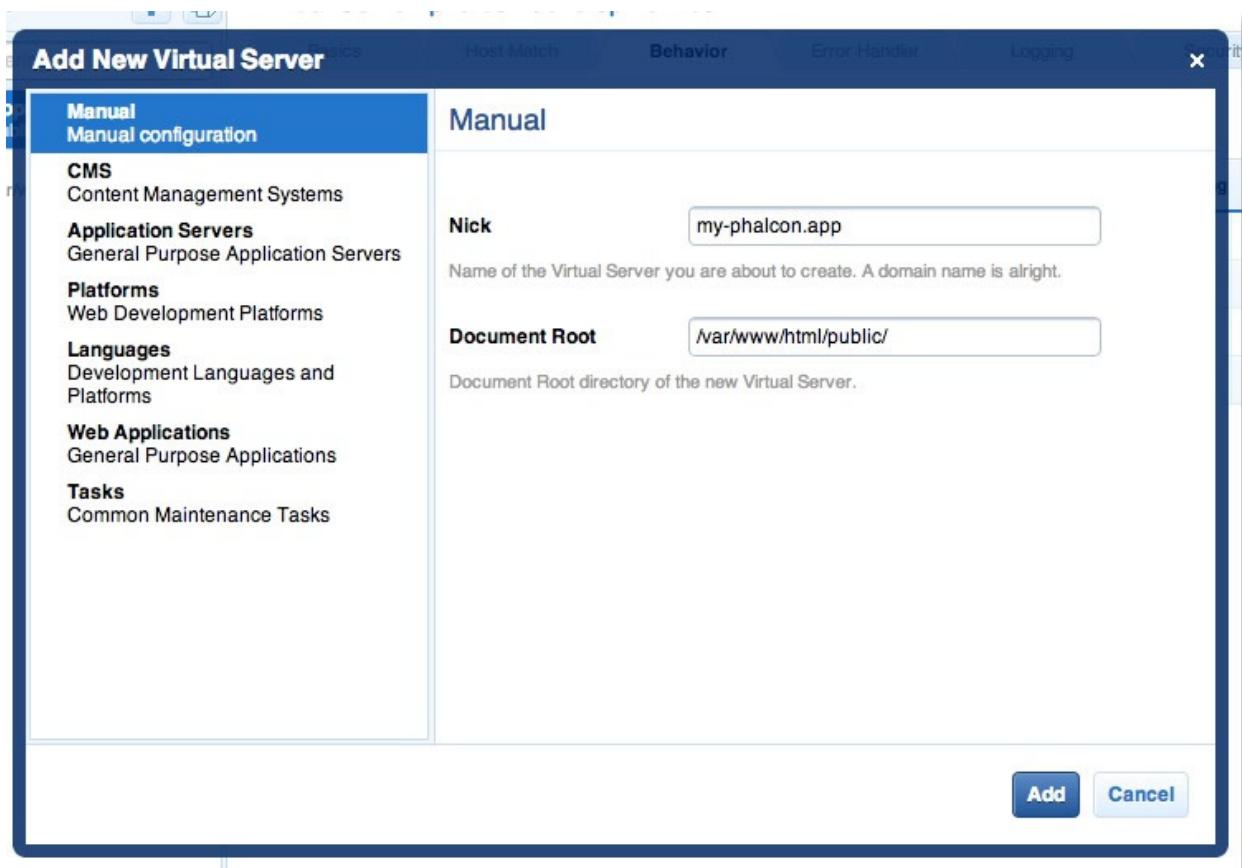
We would love to know that you are using Cherokee. Submit your domain name and it will be listed on the Cherokee Project web site.

[View list...](#) [Send your domains](#)

**Cherokee at Github**

Now you can join the [Cherokee Project development](#) team on Github! Contributing to the project has never been easier.

[Shut down Cherokee-Admin](#)



**Virtual Server: my-phalcon.app**

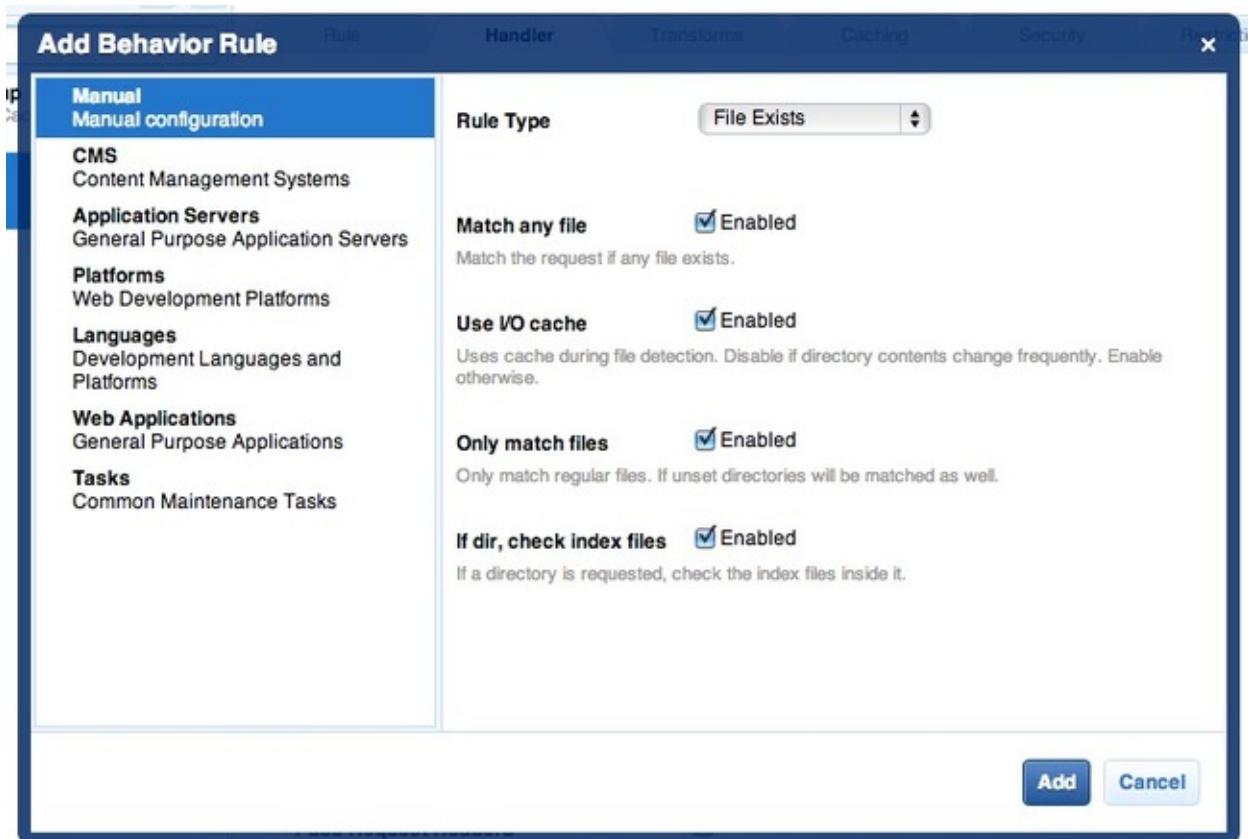
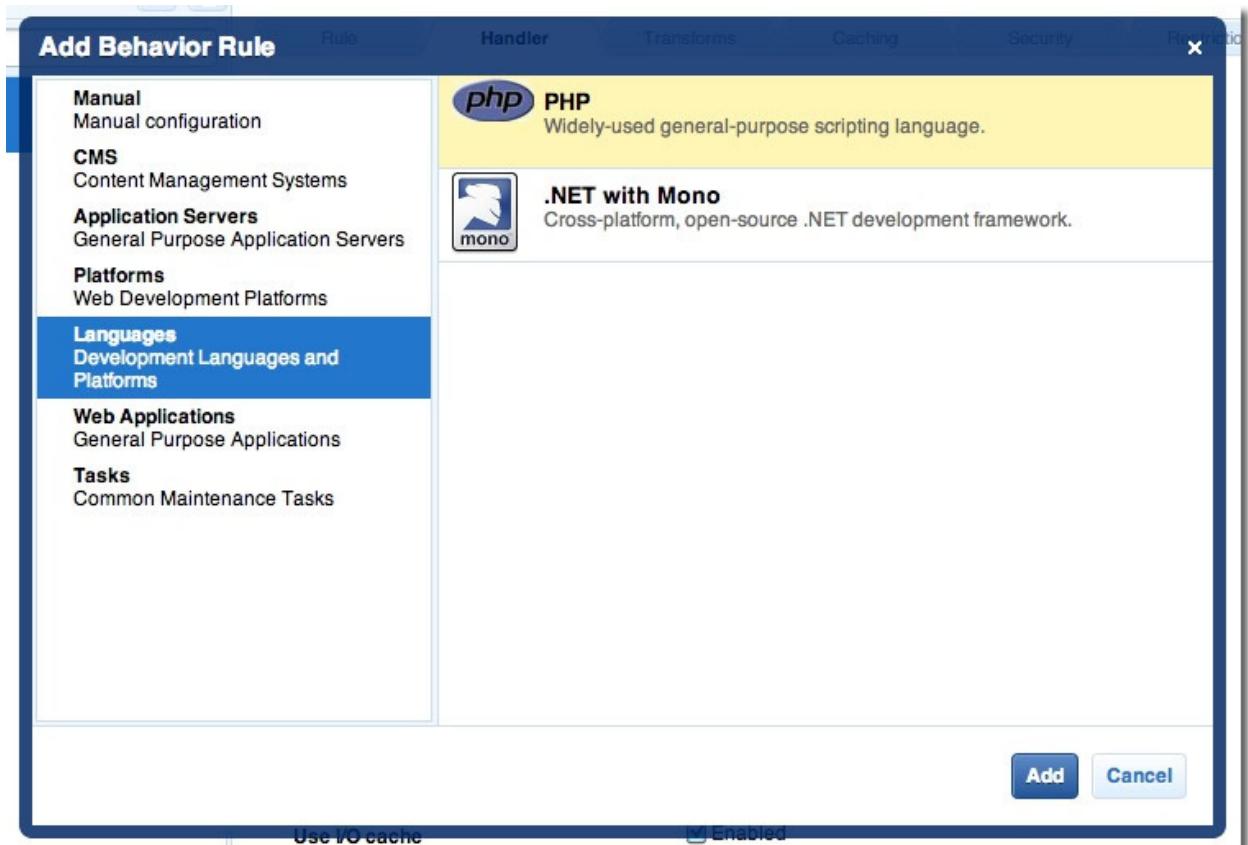
Help

Basics Host Match Behavior Error Handler Logging Security

### Behavior Rules

Match	Handler	Auth	Root	Secure	Enc	Cache	Exp	Timeout	Shaping	Log	Final
Default	List & Send									✓	✓

Rule Management

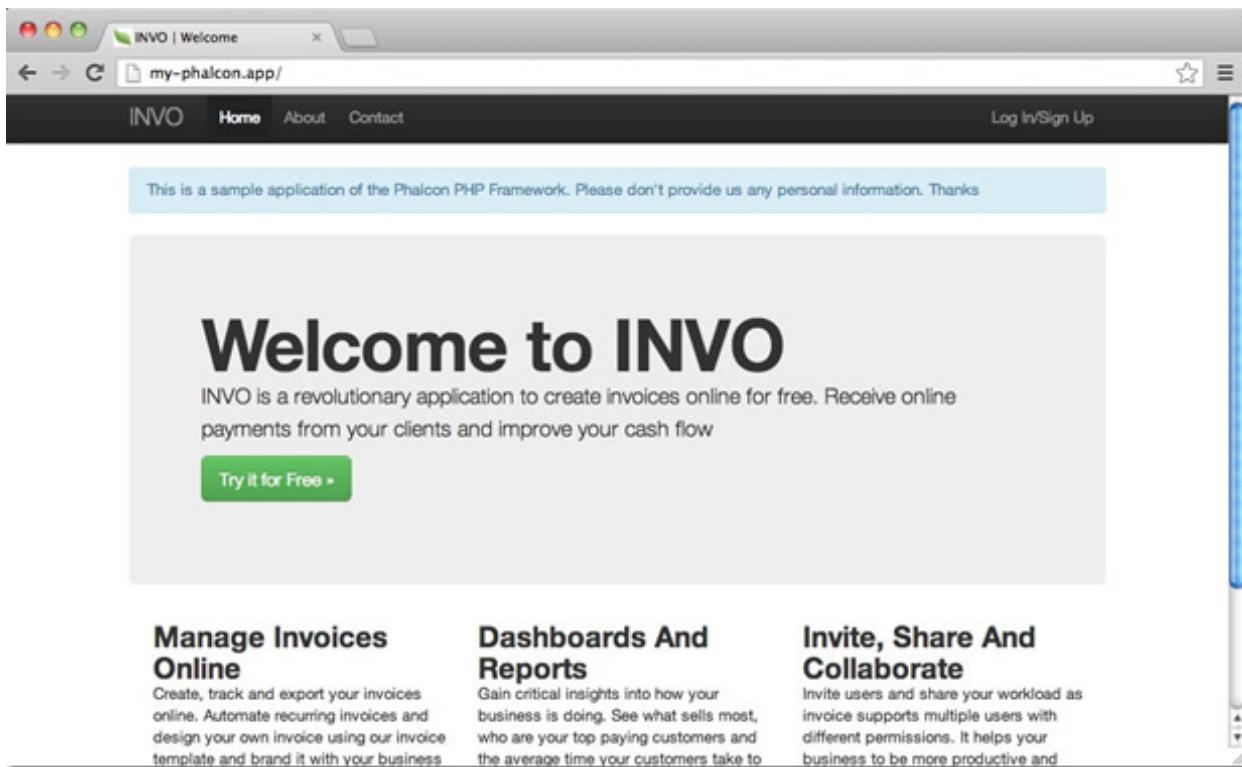


The screenshot shows the Cherokee web interface for managing a vServer named "my-phalcon.app". The "Handler" tab is selected. A rule titled "File exists" is configured with the "List & Send" handler. The "Document Root" field is set to "Optional". The "Rule Filtering" section includes a "Extensions php" rule for FastCGI, gzip, Cache, and Timeout.

The screenshot shows the Cherokee web interface for managing a vServer named "my-phalcon.app". The "Handler" tab is selected. A rule titled "Default" is configured with the "Redirection" handler. Below it, a "Rule list" table shows a single entry: Type Internal, Regular Expression `^(.*)$`, Substitution `/index.php`. The "Rule Filtering" section includes a "Extensions php" rule for FastCGI, gzip, Cache, and Timeout.

This is a zoomed-in view of the Cherokee Behavior configuration panel. It displays the following rule stack:

- Extensions php**: FastCGI, gzip, Cache, Timeout. Status: NON FINAL.
- File exists**: List & Send. Status: FINAL.
- Default**: Redirection.



```
php -S localhost:8000 -t /web_root
```

Если вы хотите перенаправлять запросы на файл index.php, добавьте файл .htrouter.php со следующим кодом:

```
<?php
if (!file_exists(__DIR__ . '/' . $_SERVER['REQUEST_URI'])) {
    $_GET['_url'] = $_SERVER['REQUEST_URI'];
}
return false;
```

и запустите сервер следующей командой:

```
php -S localhost:8000 -t /web_root .htrouter.php
```

Откройте свой браузер и перейдите по адресу <http://localhost:8000/>, чтобы убедиться, что всё работает.

## 2.4 Урок 1: Рассмотрим на примере

В этом примере рассмотрим создание приложения с простой формой регистрации “с нуля”. Также рассмотрим основные аспекты поведения фреймворка. Если вам интересна автоматическая генерация кода, посмотрите *developer tools*.

### 2.4.1 Проверка установки

Будем считать, что у вас уже установлено расширение Phalcon. Проверьте, есть ли в результатах phpinfo() секция “Phalcon” или выполните следующий код:

```
<?php print_r(get_loaded_extensions()); ?>
```

В результате вы должны увидеть Phalcon в списке:

```
Array
(
    [0] => Core
    [1] => libxml
    [2] => filter
    [3] => SPL
    [4] => standard
    [5] => phalcon
    [6] => pdo_mysql
)
```

## 2.4.2 Создание проекта

Лучше всего следовать данному руководству шаг за шагом. Полный код можно посмотреть [здесь](#).

### Структура каталогов

Phalcon не обязывает использовать определенную структуру каталогов. В виду слабой связанности фреймворка, вы можете использовать любую удобную структуру.

Для целей данного урока и для начала, мы предлагаем следующую структуру:

```
tutorial/
    app/
        controllers/
        models/
        views/
    public/
        css/
        img/
        js/
```

Обратите внимание на то, что вам не нужны директории с библиотеками, относящимися к фреймворку. Он полностью находится в памяти и все время готов к использованию.

### “Красивые” ссылки (URLs)

В этом примере будем использовать красивые УРЛ (ЧПУ). ЧПУ хороши как для SEO, так и лучше воспринимаются пользователем. Phalcon поддерживает rewrite-модули, представленные самыми распространенными веб-серверами. Вы не обязаны использовать ЧПУ в вашем приложении, вы можете с легкостью обойтись и без них.

В этом примере будем использовать rewrite модуль для Apache. Создадим несколько правил в файле `.htaccess`:

```
#!/.htaccess
<IfModule mod_rewrite.c>
    RewriteEngine on
    RewriteRule ^$ public/ [L]
    RewriteRule (.*) public/$1 [L]
</IfModule>
```

Все запросы будут перенаправлены в каталог public/ тем самым делая его корневым. Данный шаг обеспечивает закрытость внутренних файлов проекта от внешнего пользователя.

Следующий набор правил проверяет существует ли запрашиваемый файл, и если нет перенаправляет запрос index-файлу:

```
#/public/.htaccess
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-d
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php?_url=/${1} [QSA,L]
</IfModule>
```

## Bootstrap

Это первый файл, который вам необходимо создать. Это основной файл приложения, предназначенный для управления всеми его аспектами. Здесь вы можете реализовать как инициализацию компонентов приложения, так и его поведение.

Файл public/index.php содержит следующее:

```
<?php
try {

    //Register an autoloader
    $loader = new \Phalcon\Loader();
    $loader->registerDirs(array(
        '../app/controllers/',
        '../app/models/',
    ))->register();

    //Create a DI
    $di = new Phalcon\DI\FactoryDefault();

    //Setting up the view component
    $di->set('view', function(){
        $view = new \Phalcon\Mvc\View();
        $view->setViewsDir('../app/views/');
        return $view;
    });

    //Handle the request
    $application = new \Phalcon\Mvc\Application($di);

    echo $application->handle()->getContent();

} catch(\Phalcon\Exception $e) {
    echo "PhalconException: ", $e->getMessage();
}
```

## Автозагрузка

В первую очередь зарегистрируем автозагрузчик. Он будет использоваться для загрузки классов, таких как контроллеры и модели. Например мы можем зарегистрировать одну или более директорий для контроллеров, увеличив гибкость приложения. В данном примере используется компонент Phalcon\Loader.

С помощью него можно использовать разные стратегии загрузки классов, но в данном примере мы решили расположить классы в определенных директориях:

```
<?php

$loader = new \Phalcon\Loader();
$loader->registerDirs(
    array(
        '../app/controllers/',
        '../app/models/'
    )
)->register();
```

## Управление зависимостями

Важная концепция, которую стоит понять при использовании Phalcon это *dependency injection*. Это может показаться сложным, но на самом деле это очень простое и практичное.

DI представляет из себя глобальный контейнер для сервисов, необходимых нашему приложению. Каждый раз, когда фреймворку необходим какой-то компонент, он будет обращаться за ним к контейнеру используя определенное имя компонента. Так как Phalcon является слабосвязанным фреймворком, Phalcon\DI выступает в роли клея, помогающего разным компонентам прозрачно взаимодействовать друг с другом.

```
<?php

//Создание DI
$di = new Phalcon\DI\FactoryDefault();
```

*Phalcon\DI\FactoryDefault* является вариантом Phalcon\DI. Он берет на себя функции регистрации большинства компонентов из состава Phalcon, поэтому нам не придется регистрировать их вручную, один за другим. В будущем нет никакой проблемы для замены этого сервиса своим.

На следующем шаге мы регистрируем сервис ‘view’, который указывает на папку с файлами ‘view’ (вьюхи). Т.к. данные файлы не относятся к классам, они не могут быть подгружены автoloадером.

Существует несколько путей для регистрации сервисов, но в нашем примере мы используем анонимную функцию:

```
<?php

//Setting up the view component
$di->set('view', function(){
    $view = new \Phalcon\Mvc\View();
    $view->setViewsDir('../app/views/');
    return $view;
});
```

На последнем этапе мы используем *Phalcon\Mvc\Application*. Данная компонента служит для инициализации окружения входящих запросов, их перенаправления и обслуживания относящихся к ним действий. После отработки всех доступных действий, компонента возвращает полученные результаты.

```
<?php

$application = new \Phalcon\Mvc\Application($di);

echo $application->handle()->getContent();
```

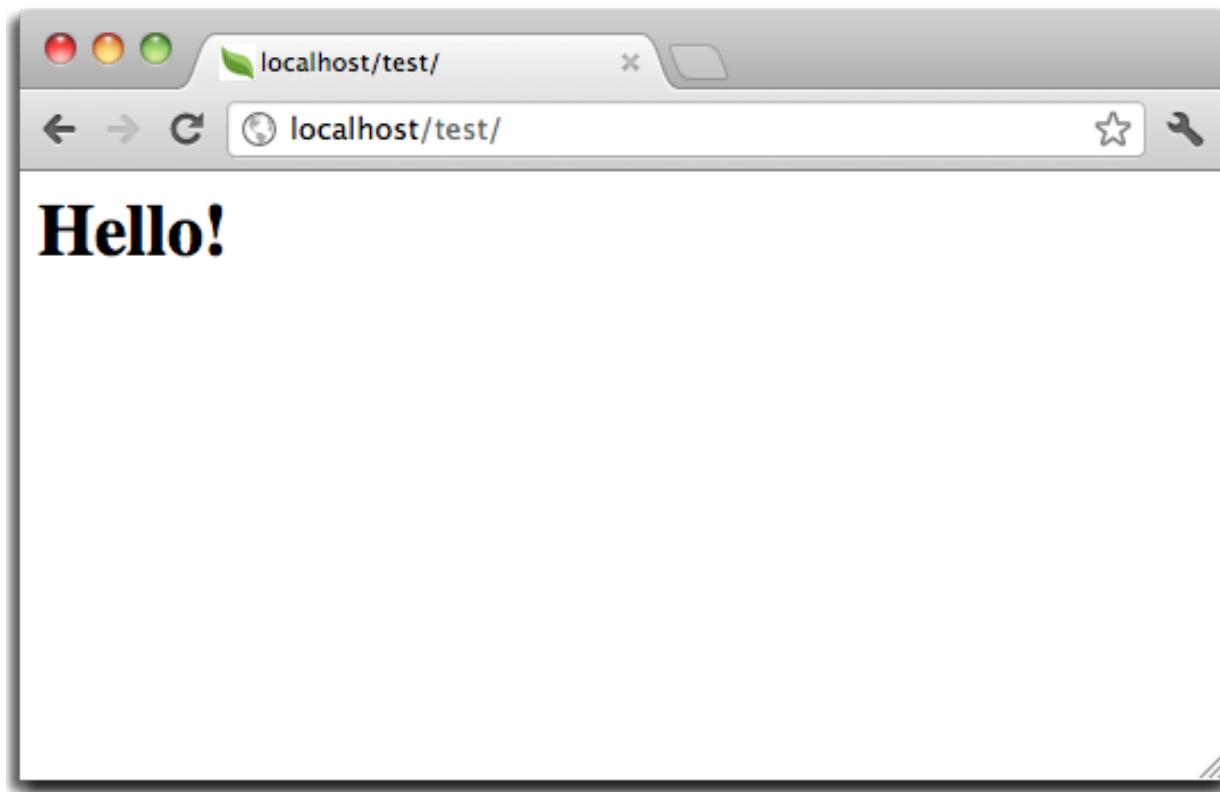
Как можно увидеть, файл инициализации очень короткий, нам нет необходимости подключать какие-либо дополнительные файлы. Таким образом, мы настроили гибкую структуру MVC-приложения менее чем за 30 строк кода.

## Создание контроллера

По умолчанию Phalcon будет искать контроллер с именем “Index”. Как и во многих других фреймворках, он является исходной точкой, когда ни один другой контроллер или действие не были запрошены. Наш контроллер по умолчанию (`app/controllers/IndexController.php`) выглядит так:

```
<?php  
  
class IndexController extends \Phalcon\Mvc\Controller  
{  
  
    public function indexAction()  
    {  
        echo "<h1>Привет!</h1>";  
    }  
  
}
```

Классы контроллеров должны заканчиваться на “Controller”, чтобы автозагрузчик смог загрузить их, а их действия должны заканчиваться на “Action” по той же причине. Теперь можно открыть браузер и увидеть результат:



Удача! Phalcon моментально отображает нашу простенькую страницу!

## Отправка результатов для просмотра

Отображение вывода напрямую из контроллера временами может быть хорошей идеей (например, когда нужно отправить JSON), но не всегда разумно, и сторонники шаблона MVC это подтверждают. Гораздо правильнее использовать отдельные файлы представлений (view). Phalcon ищет файл представления с именем, совпадающим с именем действия внутри папки, совпадающей с именем последнего запущенного контроллера. В нашем случае это будет выглядеть так (app/views/index/index.phtml):

```
<?php echo "<h1>Привет!</h1>";
```

В нашем контроллере (app/controllers/IndexController.php) теперь пустое определение действия:

```
<?php
```

```
class IndexController extends \Phalcon\Mvc\Controller
{
    public function indexAction()
    {
    }
}
```

Вывод браузера останется прежним. Когда действие завершит свою работу, будет автоматически создан статический компонент *Phalcon\ Mvc\ View*. Узнать больше о представлениях можно [здесь](#).

## Проектирование формы регистрации

Давайте теперь изменим файл представления index.phtml, добавив ссылку на новый контроллер "signup". Идея проста - позволить пользователям регистрироваться в нашем приложении.

```
<?php
echo "<h1>Привет!</h1>";
echo Phalcon\Tag::linkTo("signup", "Регистрируйся!");
```

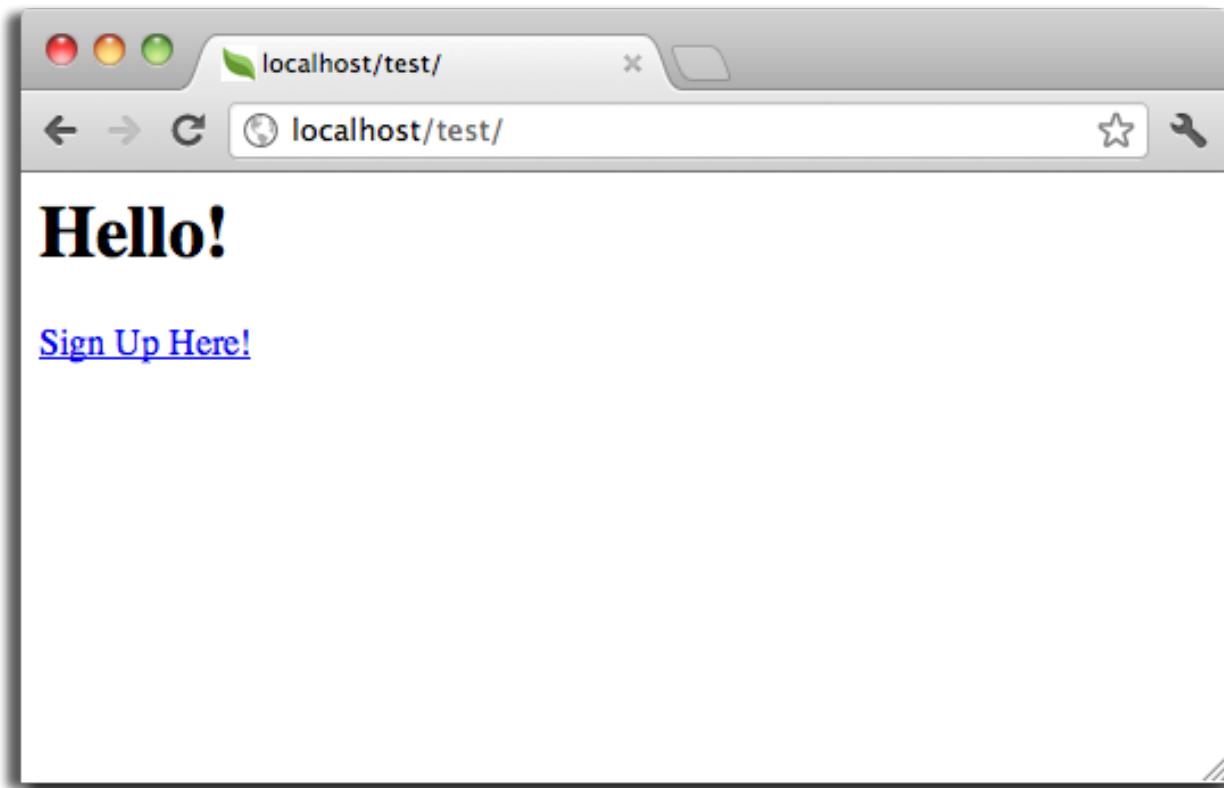
Сгенерированный код HTML будет выводить тэг “” , указывающий на наш новый контроллер:

```
<h1>Привет!</h1> <a href="/test/signup">Регистрируйся!</a>
```

Для генерации тэга мы воспользовались встроенным классом *Phalcon\ Tag*. Это служебный класс, позволяющий конструировать HTML-разметку в Phalcon-подобном стиле. Более подробно можно [узнать здесь](#).

Контроллер Signup сейчас очень похож на предыдущий контроллер и выглядит так (app/controllers/SignupController.php):

```
<?php
class SignupController extends \Phalcon\Mvc\Controller
{
    public function indexAction()
    {
    }
}
```



}

Пустое действие index говорит нам о том, что будет использоваться одноименный файл представления с нашей формой для регистрации (app/views/signup/index.phtml):

```
<?php use Phalcon\Tag; ?>

<h2>Sign using this form</h2>

<?php echo Tag::form("signup/register"); ?>

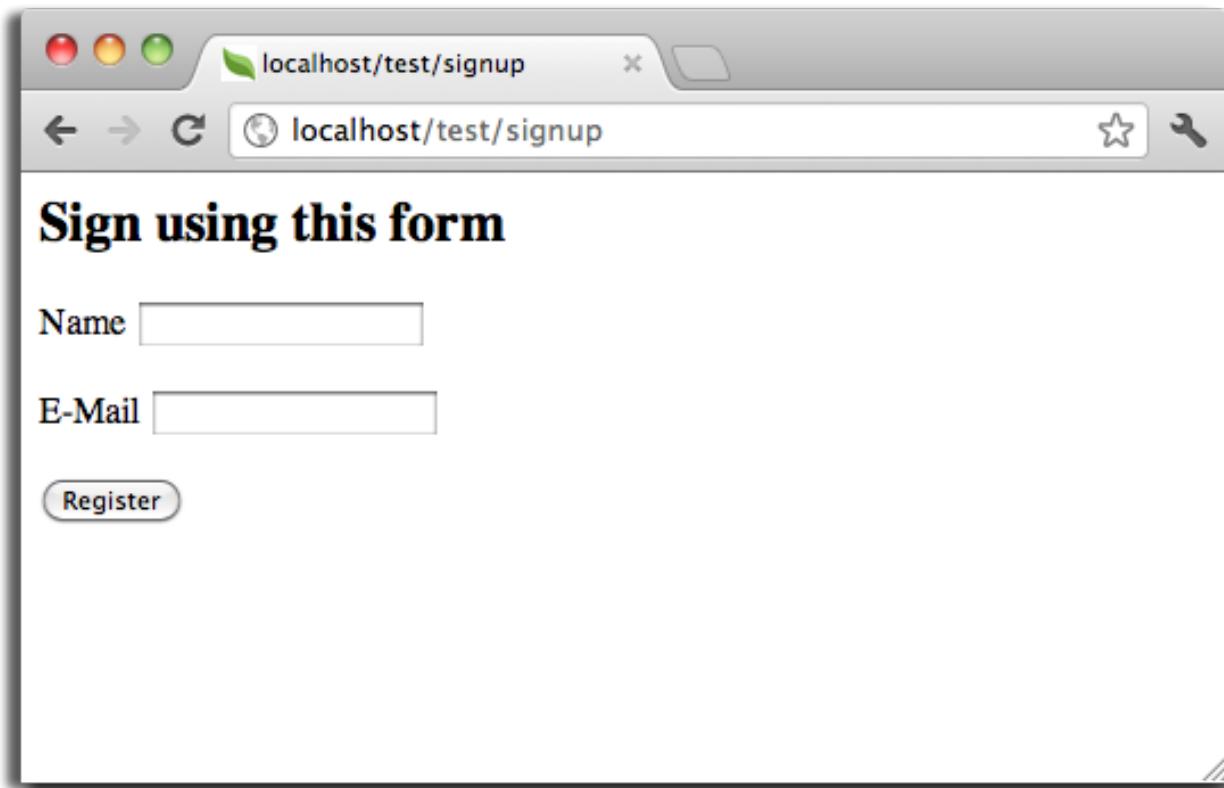
<p>
    <label for="name">Имя</label>
    <?php echo Tag::textField("name") ?>
</p>

<p>
    <label for="email">E-Mail</label>
    <?php echo Tag::textField("email") ?>
</p>

<p>
    <?php echo Tag::submitButton("Регистрация") ?>
</p>

</form>
```

В браузере это будет выглядеть так:



Класс `Phalcon\Tag` также содержит полезные методы для работы с формами.

Метод `Phalcon\Tag::form` принимает единственный аргумент, например, относительный идентификатор контроллер/действие приложения.

При нажатии на кнопку "Регистрация" видим исключение, вызванное фреймворком. Оно говорит нам о том, что отсутствует действие "register" нашего контроллера "signup":

`PhalconException: Action "register" was not found on controller "signup"`

Не будем испытывать судьбу и реализуем данный метод:

```
<?php

class SignupController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function registerAction()
    {

    }
}
```

Снова жмем на кнопку "Регистрация" и видим пустую страницу. Поля name и email, введенные пользователем, должны сохраниться в базе данных. Следуя традиции MVC, все взаимодействие с БД должно

вестись через модели, следуя традициям ООП-стиля.

## Создание модели

Phalcon содержит первую ORM для PHP, полностью написанную на языке С. Вместо усложнения процесса разработки, он упрощает ее!

Мы должны связать таблицу в нашей базе данных перед созданием нашей первой модели. Простейшая таблица для регистрации пользователей приведена ниже:

```
CREATE TABLE `users` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `name` varchar(70) NOT NULL,
    `email` varchar(70) NOT NULL,
    PRIMARY KEY (`id`)
);
```

Файлы моделей должны находиться в папке app/models (app/models/Users.php). Модель, представляющая таблицу “users”, выглядит следующим образом:

```
<?php

class Users extends \Phalcon\Mvc\Model
{
```

}

## Настройка соединения с базой данных

Для использования базы данных и получения к ней доступа через наши модели, нам необходимо указать настройки в процессе инициализации нашего приложения. Соединение с базой данных это всего лишь еще один сервис в нашем сервис-локаторе:

```
<?php

try {

    // Регистрация автозагрузчика
    $loader = new \Phalcon\Loader();
    $loader->registerDirs(array(
        '../app/controllers/',
        '../app/models/',
    ))->register();

    // Создание DI
    $di = new Phalcon\DI\FactoryDefault();

    // Настраиваем сервис для работы с БД
    $di->set('db', function(){
        return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
            "host" => "localhost",
            "username" => "root",
            "password" => "secret",
            "dbname" => "test_db"
        ));
    });
}
```

```
// Настраиваем компонент View
$di->set('view', function(){
    $view = new \Phalcon\Mvc\View();
    $view->setViewsDir('../app/views/');
    return $view;
});

// Обработка запроса
$application = new \Phalcon\Mvc\Application($di);

echo $application->handle()->getContent();

} catch(Exception $e) {
    echo "PhalconException: ", $e->getMessage();
}
```

При правильных настройках подключения наши модели готовы к работе и взаимодействию с остальными частями приложения.

### Сохранение данных при работе с моделями

Следующим шагом будет обработка данных нашей формы регистрации и сохранение их в таблице базы данных.

```
<?php

class SignupController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function registerAction()
    {

        $user = new Users();

        // Сохраняем и проверяем на наличие ошибок
        $success = $user->save($this->request->getPost(), array('name', 'email'));

        if ($success) {
            echo "Спасибо за регистрацию!";
        } else {
            echo "К сожалению, возникли следующие проблемы: ";
            foreach ($user->getMessages() as $message) {
                echo $message->getMessage(), "<br/>";
            }
        }

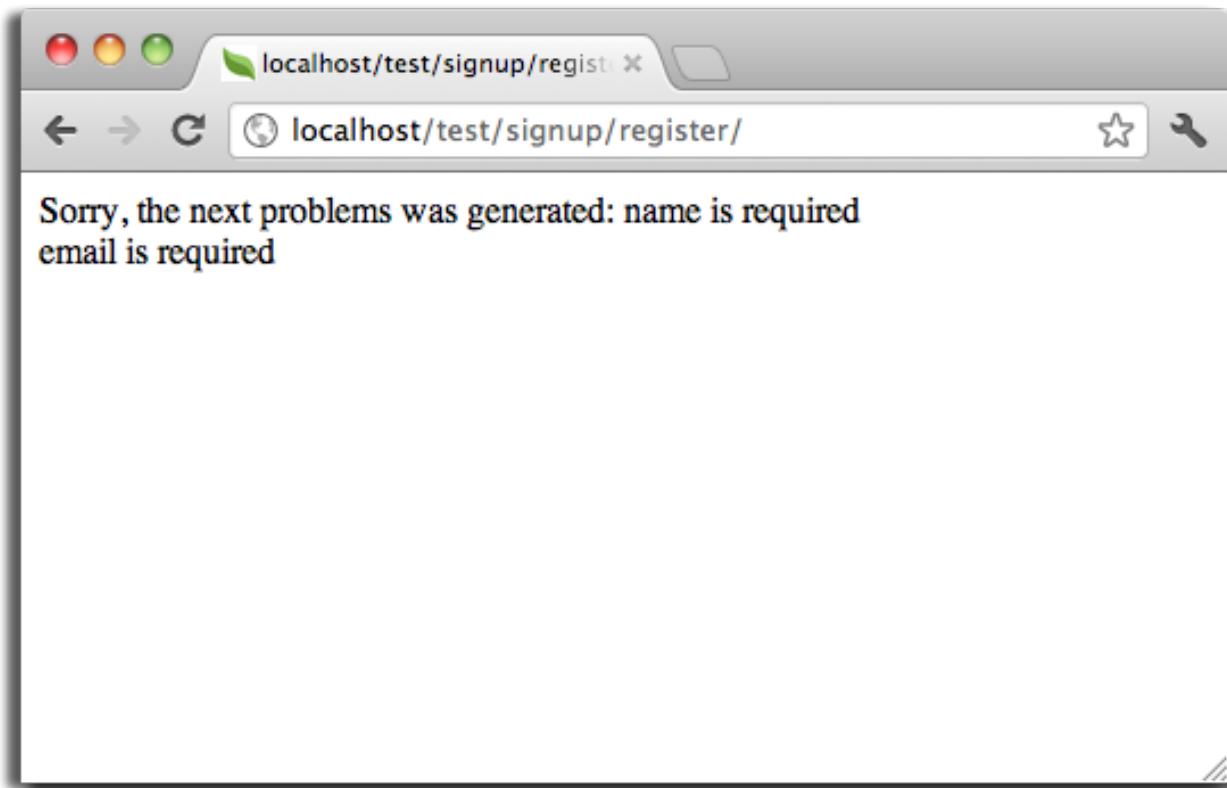
        $this->view->disable();
    }
}
```

В действии ‘register’ мы создаем экземпляр модели Users, отвечающий за записи пользователей. Пуб-

личные свойства класса указывают на их одноименные названия полей в таблице базы данных. Установка необходимых значений нашей модели и вызов метода `save()` приводит к сохранению этих данных в базе данных. Метод `save()` возвращает булево значение указывающее, успешно ли были сохранены данные в таблице или нет (`true` и `false`, соответственно).

ORM автоматически экранирует ввод для предотвращения SQL-инъекций, так что мы можем передавать массив `$_POST` напрямую методу `save()`.

Для полей, у которых установлен параметр `not null` (обязательные), необходима дополнительная валидация. Без нее мы получим что-то вроде этого:



### 2.4.3 Заключение

На этом очень простом руководстве можно увидеть, как просто начать создавать приложения с помощью Phalcon. То, что Phalcon является расширением, никак не влияет на сложность разработки и доступные возможности. Продолжайте читать данное руководство для изучения новых возможностей, которые предоставляет Phalcon!

### 2.4.4 Примеры приложений

Можно ознакомиться с более развернутыми примерами приложений, написанных с помощью Phalcon:

- [INVO application](#): Приложение для создания счетов. Позволяет редактировать продукты, компании, типы продуктов и др.
- [PHP Alternative website](#): Мультиязычное приложение с продвинутым роутингом.

- **Album O'Rama**: Витрина музыкальных альбомов. Обработка больших объемов данных с помощью диалекта *PHQL* и шаблонизатора *Volt*
- **Phosphorum**: Простой форум

## 2.5 Tutorial 2: Explaining INVO

In this second tutorial, we'll explain a more complete application in order to deepen the development with Phalcon. INVO is one of the applications we have created as samples. INVO is a small website that allows their users to generate invoices, and do other tasks as manage their customers and products. You can clone its code from [Github](#).

Also, INVO was made with [Twitter Bootstrap](#) as client-side framework. Although the application does not generate invoices still it serves as an example to understand how the framework works.

### 2.5.1 Project Structure

Once you clone the project in your document root you'll see the following structure:

```
invo/
  app/
    app/config/
    app/controllers/
    app/library/
    app/models/
    app/plugins/
    app/views/
  public/
    public/bootstrap/
    public/css/
    public/js/
  schemas/
```

As you know, Phalcon does not impose a particular file structure for application development. This project provides a simple MVC structure and a public document root.

Once you open the application in your browser <http://localhost/invo> you'll something like this:

The application is divided in two parts, a frontend, that is a public part where visitors can receive information about INVO and request contact information. The second part is the backend, an administrative area where a registered user can manage his/her products and customers.

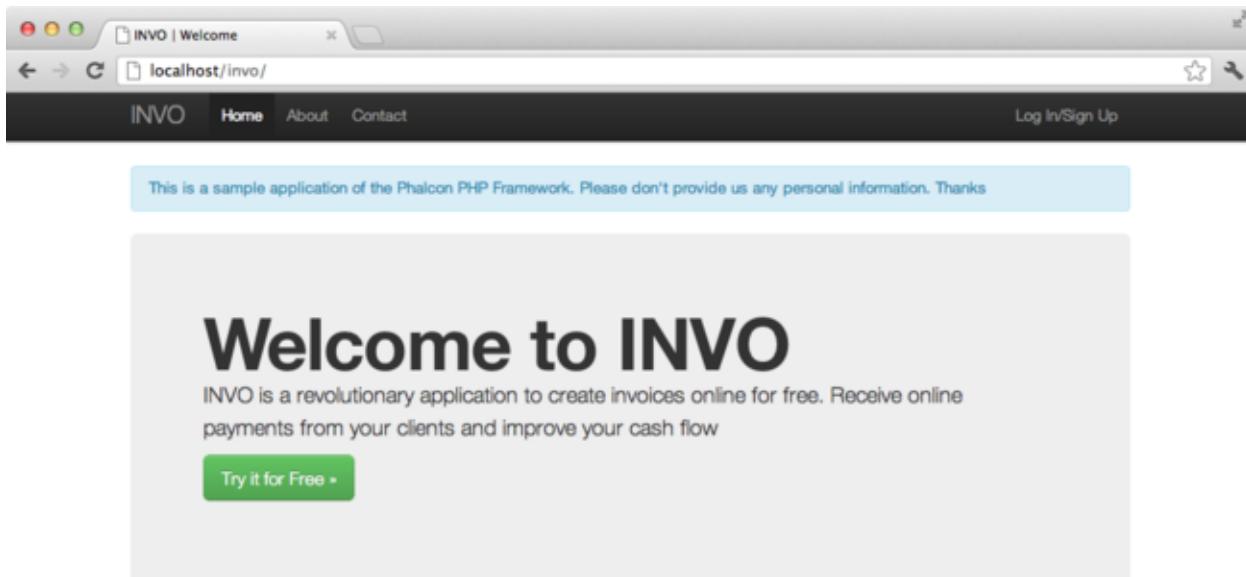
### 2.5.2 Routing

INVO uses the standard route that is built-in with the Router component. These routes matches the following pattern: `/:controller/:action/:params`. This means that the first part of an URI is the controller, the second the action and the rest are the parameters.

The following route `/session/register` executes the controller `SessionController` and its action `registerAction`.

### 2.5.3 Configuration

INVO has a configuration file that sets general parameters in the application. This file is read in the first lines of the bootstrap file (`public/index.php`):



## Manage Invoices Online

Create, track and export your invoices online. Automate recurring invoices and design your own invoice using our invoice template and brand it with your business

## Dashboards And Reports

Gain critical insights into how your business is doing. See what sells most, who are your top paying customers and the average time your customers take to

## Invite, Share And Collaborate

Invite users and share your workload as invoice supports multiple users with different permissions. It helps your business to be more productive and

```
<?php
```

```
//Read the configuration
$config = new Phalcon\Config\Adapter\Ini('..../app/config/config.ini');
```

*Phalcon|Config* allows us to manipulate the file in an object-oriented way. The configuration file contains the following settings:

```
[database]
host      = localhost
username  = root
password  = secret
name      = invo

[application]
controllersDir = ..../app/controllers/
modelsDir     = ..../app/models/
viewsDir      = ..../app/views/
pluginsDir    = ..../app/plugins/
libraryDir    = ..../app/library/
baseUri       = /invo/

;[metadata]
;adapter = "Apc"
;suffix = my-suffix
;lifetime = 3600
```

Phalcon hasn't any pre-defined convention settings. Sections help us to organize the options as appropriate. In this file there are three sections to be used later.

## 2.5.4 Autoloaders

A second part that appears in the bootstrap file (public/index.php) is the autoloader. The autoloader registers a set of directories where the application will look for the classes that it eventually will need.

```
<?php

$loader = new \Phalcon\Loader();

$loader->registerDirs(
    array(
        $config->application->controllersDir,
        $config->application->pluginsDir,
        $config->application->libraryDir,
        $config->application->modelsDir,
    )
)->register();
```

Note that what has been done is registering the directories that were defined in the configuration file. The only directory that is not registered is the viewsDir, because it contains no classes but html + php files.

## 2.5.5 Handling the Request

Let's go much further, at the end of the file, the request is finally handled by Phalcon\Mvc\Application, this class initializes and executes all the necessary to make the application run:

```
<?php

$app = new \Phalcon\Mvc\Application($di);

echo $app->handle()->getContent();
```

## 2.5.6 Dependency Injection

Look at the first line of the code block above, the variable \$app is receiving another variable \$di in its constructor. What is the purpose of that variable? Phalcon is a highly decoupled framework, so we need a component that acts as glue to make everything work together. That component is Phalcon\DI. It is a service container that also performs dependency injection, instantiating all components, as they are needed by the application.

There are many ways of registering services in the container. In INVO most services have been registered using anonymous functions. Thanks to this, the objects are instantiated in a lazy way, reducing the resources needed by the application.

For instance, in the following excerpt, the session service is registered, the anonymous function will only be called when the application requires access to the session data:

```
<?php

//Start the session the first time when some component request the session service
$di->set('session', function() {
    $session = new Phalcon\Session\Adapter\Files();
    $session->start();
    return $session;
});
```

Here, we have the freedom to change the adapter, perform additional initialization and much more. Note that the service was registered using the name “session”. This is a convention that will allow the framework to identify the active service in the services container.

A request can use many services, register each service one to one can be a cumbersome task. For that reason, the framework provides a variant of Phalcon\DI called Phalcon\DI\FactoryDefault whose task is to register all services providing a full-stack framework.

```
<?php
```

```
// The FactoryDefault Dependency Injector automatically registers the
// right services providing a full stack framework
$di = new \Phalcon\DI\FactoryDefault();
```

It registers the majority of services with components provided by the framework as standard. If we need to override the definition of some service we could just set it again as we did above with “session”. This is the reason for the existence of the variable \$di.

### 2.5.7 Log into the Application

“Log in” will allow us to work on backend controllers. The separation between backend’s controllers and the frontend ones is only logical. All controllers are located in the same directory (app/controllers/).

To enter into the system, we must have a valid username and password. Users are stored in the table “users” in the database “invo”.

Before we can start a session, we need to configure the connection to the database in the application. A service called “db” is set up in the service container with that information. As with the autoloader, this time we are also taking parameters from the configuration file in order to configure a service:

```
<?php
```

```
// Database connection is created based on the parameters defined in the configuration file
$di->set('db', function() use ($config) {
    return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
        "host" => $config->database->host,
        "username" => $config->database->username,
        "password" => $config->database->password,
        "dbname" => $config->database->name
    ));
});
```

Here, we return an instance of the MySQL connection adapter. If needed, you could do extra actions such as adding a logger, a profiler or change the adapter, setting up it as you want.

Back then, the following simple form (app/views/session/index.phtml) requests the logon information. We’ve removed some HTML code to make the example more concise:

```
<?php echo $this->tag->form('session/start') ?>

<label for="email">Username/Email</label>
<?php echo $this->tag->textField(array("email", "size" => "30")) ?>

<label for="password">Password</label>
<?php echo $this->tag->passwordField(array("password", "size" => "30")) ?>

<?php echo $this->tag->submitButton(array('Login')) ?>

</form>
```

The SessionController::startAction (app/controllers/SessionController.phtml) has the task of validate the data entered checking for a valid user in the database:

```
<?php

class SessionController extends ControllerBase
{

    // ...

    private function _registerSession($user)
    {
        $this->session->set('auth', array(
            'id' => $user->id,
            'name' => $user->name
        ));
    }

    public function startAction()
    {
        if ($this->request->isPost()) {

            //Receiving the variables sent by POST
            $email = $this->request->getPost('email', 'email');
            $password = $this->request->getPost('password');

            $password = sha1($password);

            //Find for the user in the database
            $user = Users::findFirst(array(
                "email = :email: AND password = :password: AND active = 'Y'",
                "bind" => array('email' => $email, 'password' => $password)
            ));
            if ($user != false) {

                $this->_registerSession($user);

                $this->flash->success('Welcome ' . $user->name);

                //Forward to the 'invoices' controller if the user is valid
                return $this->dispatcher->forward(array(
                    'controller' => 'invoices',
                    'action' => 'index'
                ));
            }
            $this->flash->error('Wrong email/password');
        }

        //Forward to the login form again
        return $this->dispatcher->forward(array(
            'controller' => 'session',
            'action' => 'index'
        ));
    }
}
```

For simplicity, we have used “`sha1`” to store the password hashes in the database, however, this algorithm is not recommended in real applications, use ” `bcrypt`” instead.

Note that multiple public attributes are accessed in the controller like: `$this->flash`, `$this->request` or `$this->session`. These are services defined in services container from earlier. When they’re accessed the first time, are injected as part of the controller.

These services are shared, which means that we are always accessing the same instance regardless of the place where we invoke them.

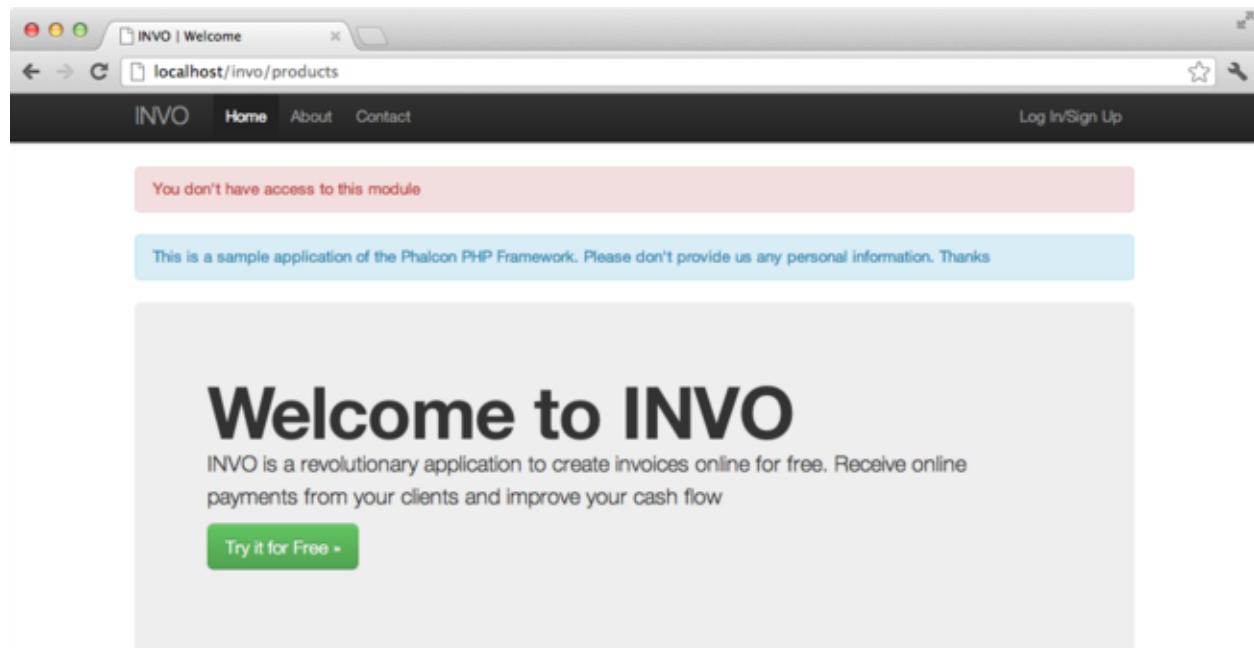
For instance, here we invoke the “`session`” service and then we store the user identity in the variable “`auth`”:

```
<?php
```

```
$this->session->set('auth', array(
    'id' => $user->id,
    'name' => $user->name
));
```

## 2.5.8 Securing the Backend

The backend is a private area where only registered users have access. Therefore, it is necessary to check that only registered users have access to these controllers. If you aren’t logged in the application and you try to access, for example, the products controller (that is private) you will see a screen like this:



Every time someone attempts to access any controller/action, the application verifies that the current role (in session) has access to it, otherwise it displays a message like the above and forwards the flow to the home page.

Now let's find out how the application accomplishes this. The first thing to know is that there is a component called *Dispatcher*. It is informed about the route found by the *Routing* component. Then, it is responsible for loading the appropriate controller and execute the corresponding action method.

Normally, the framework creates the Dispatcher automatically. In our case, we want to perform a verification before executing the required action, checking if the user has access to it or not. To achieve this, we have replaced the component by creating a function in the bootstrap:

```
<?php

$di->set('dispatcher', function() use ($di) {
    $dispatcher = new Phalcon\Mvc\Dispatcher();
    return $dispatcher;
});
```

We now have total control over the Dispatcher used in the application. Many components in the framework trigger events that allow us to modify their internal flow of operation. As the dependency Injector component acts as glue for components, a new component called *EventsManager* aids us to intercept the events produced by a component routing the events to listeners.

## Events Management

A *EventsManager* allows us to attach listeners to a particular type of event. The type that interest us now is "dispatch", the following code filters all events produced by the Dispatcher:

```
<?php

$di->set('dispatcher', function() use ($di) {

    //Obtain the standard eventsManager from the DI
    $eventsManager = $di->getShared('eventsManager');

    //Instantiate the Security plugin
    $security = new Security($di);

    //Listen for events produced in the dispatcher using the Security plugin
    $eventsManager->attach('dispatch', $security);

    $dispatcher = new Phalcon\Mvc\Dispatcher();

    //Bind the EventsManager to the Dispatcher
    $dispatcher->setEventsManager($eventsManager);

    return $dispatcher;
});
```

The Security plugin is a class located at (app/plugins/Security.php). This class implements the method "beforeExecuteRoute". This is the same name as one of the events produced in the Dispatcher:

```
<?php

use Phalcon\Events\Event,
    Phalcon\Mvc\Dispatcher,
    Phalcon\Mvc\User\Plugin;

class Security extends Plugin
{
```

```
// ...

public function beforeExecuteRoute(Event $event, Dispatcher $dispatcher)
{
    // ...
}

}
```

The hooks events always receive a first parameter that contains contextual information of the event produced (\$event) and a second one that is the object that produced the event itself (\$dispatcher). It is not mandatory that plugins extend the class Phalcon\Mvc\User\Plugin, but by doing this they gain easier access to the services available in the application.

Now, we're verifying the role in the current session, checking if he/she has access using the ACL list. If he/she does not have access we redirect him/her to the home screen as explained before:

```
<?php

use Phalcon\Events\Event,
    Phalcon\Mvc\Dispatcher,
    Phalcon\Mvc\User\Plugin;

class Security extends Plugin
{

    // ...

    public function beforeExecuteRoute(Event $event, Dispatcher $dispatcher)
    {

        //Check whether the "auth" variable exists in session to define the active role
        $auth = $this->session->get('auth');
        if (!$auth) {
            $role = 'Guests';
        } else {
            $role = 'Users';
        }

        //Take the active controller/action from the dispatcher
        $controller = $dispatcher->getControllerName();
        $action = $dispatcher->getActionName();

        //Obtain the ACL list
        $acl = $this->_getAcl();

        //Check if the Role have access to the controller (resource)
        $allowed = $acl->isAllowed($role, $controller, $action);
        if ($allowed != Phalcon\Acl::ALLOW) {

            //If he doesn't have access forward him to the index controller
            $this->flash->error("You don't have access to this module");
            $dispatcher->forward(
                array(
                    'controller' => 'index',
                    'action' => 'index'
                )
            );
        }
    }
}
```

```
//Returning "false" we tell to the dispatcher to stop the current operation
return false;
}

}

}
```

## Providing an ACL list

In the above example we have obtained the ACL using the method `$this->_getAcl()`. This method is also implemented in the Plugin. Now we are going to explain step-by-step how we built the access control list (ACL):

```
<?php

//Create the ACL
$acl = new Phalcon\Acl\Adapter\Memory();

//The default action is DENY access
$acl->setDefaultAction(Phalcon\Acl::DENY);

//Register two roles, Users is registered users
//and guests are users without a defined identity
$roles = array(
    'users' => new Phalcon\Acl\Role('Users'),
    'guests' => new Phalcon\Acl\Role('Guests')
);
foreach ($roles as $role) {
    $acl->addRole($role);
}
```

Now we define the resources for each area respectively. Controller names are resources and their actions are accesses for the resources:

```
<?php

//Private area resources (backend)
$privateResources = array(
    'companies' => array('index', 'search', 'new', 'edit', 'save', 'create', 'delete'),
    'products' => array('index', 'search', 'new', 'edit', 'save', 'create', 'delete'),
    'producttypes' => array('index', 'search', 'new', 'edit', 'save', 'create', 'delete'),
    'invoices' => array('index', 'profile')
);
foreach ($privateResources as $resource => $actions) {
    $acl->addResource(new Phalcon\Acl\Resource($resource), $actions);
}

//Public area resources (frontend)
$publicResources = array(
    'index' => array('index'),
    'about' => array('index'),
    'session' => array('index', 'register', 'start', 'end'),
    'contact' => array('index', 'send')
);
foreach ($publicResources as $resource => $actions) {
    $acl->addResource(new Phalcon\Acl\Resource($resource), $actions);
```

```
}
```

The ACL now have knowledge of the existing controllers and their related actions. Role “Users” has access to all the resources of both frontend and backend. The role “Guests” only has access to the public area:

```
<?php

//Grant access to public areas to both users and guests
foreach ($roles as $role) {
    foreach ($publicResources as $resource => $actions) {
        $acl->allow($role->getName(), $resource, '*');
    }
}

//Grant access to private area only to role Users
foreach ($privateResources as $resource => $actions) {
    foreach ($actions as $action) {
        $acl->allow('Users', $resource, $action);
    }
}
```

Hooray!, the ACL is now complete.

### 2.5.9 User Components

All the UI elements and visual style of the application has been achieved mostly through Twitter Bootstrap. Some elements, such as the navigation bar changes according to the state of the application. For example, in the upper right corner, the link “Log in / Sign Up” changes to “Log out” if an user is logged into the application.

This part of the application is implemented in the component “Elements” (app/library/Elements.php).

```
<?php

use Phalcon\Mvc\User\Component;

class Elements extends Component
{

    public function getMenu()
    {
        //...
    }

    public function getTabs()
    {
        //...
    }
}
```

This class extends the Phalcon\Mvc\User\Component, it is not imposed to extend a component with this class, but it helps to get access more quickly to the application services. Now, we register this class in the services container:

```
<?php

//Register an user component
```

```
$di->set('elements', function(){
    return new Elements();
});
```

As controllers, plugins or components within a view, this component also has access to the services registered in the container and by just accessing an attribute with the same name as a previously registered service:

```
<div class="navbar navbar-fixed-top">
    <div class="navbar-inner">
        <div class="container">
            <a class="btn btn-navbar" data-toggle="collapse" data-target=".nav-collapse">
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </a>
            <a class="brand" href="#">INVO</a>
            <?php echo $this->elements->getMenu() ?>
        </div>
    </div>
</div>

<div class="container">
    <?php echo $this->getContent() ?>
    <hr>
    <footer>
        <p>&copy; Company 2012</p>
    </footer>
</div>
```

The important part is:

```
<?php echo $this->elements->getMenu() ?>
```

## 2.5.10 Working with the CRUD

Most options that manipulate data (companies, products and types of products), were developed using a basic and common **CRUD** (Create, Read, Update and Delete). Each CRUD contains the following files:

```
invo/
    app/
        app/controllers/
            ProductsController.php
        app/models/
            Products.php
        app/views/
            products/
                edit.phtml
                index.phtml
                new.phtml
                search.phtml
```

Each controller has the following actions:

```
<?php

class ProductsController extends ControllerBase
{
```

```
/**
 * The start action, it shows the "search" view
 */
public function indexAction()
{
    //...
}

/**
 * Execute the "search" based on the criteria sent from the "index"
 * Returning a paginator for the results
 */
public function searchAction()
{
    //...
}

/**
 * Shows the view to create a "new" product
 */
public function newAction()
{
    //...
}

/**
 * Shows the view to "edit" an existing product
 */
public function editAction()
{
    //...
}

/**
 * Creates a product based on the data entered in the "new" action
 */
public function createAction()
{
    //...
}

/**
 * Updates a product based on the data entered in the "edit" action
 */
public function saveAction()
{
    //...
}

/**
 * Deletes an existing product
 */
public function deleteAction($id)
{
    //...
}

}
```

## The Search Form

Every CRUD starts with a search form. This form shows each field that has the table (products), allowing the user creating a search criteria from any field. Table “products” has a relationship to the table “products\_types”. In this case, we previously queried the records in this table in order to facilitate the search by that field:

```
<?php

/**
 * The start action, it shows the "search" view
 */
public function indexAction()
{
    $this->persistent->searchParams = null;
    $this->view->productTypes = ProductTypes::find();
}
```

All the “product types” are queried and passed to the view as a local variable “productTypes”. Then, in the view (app/views/index.phtml) we show a “select” tag filled with those results:

```
<div>
    <label for="product_types_id">Product Type</label>
    <?php echo $this->tag->select(array(
        "product_types_id",
        $productTypes,
        "using" => array("id", "name"),
        "useDummy" => true
    )) ?>
</div>
```

Note that \$productTypes contains the data necessary to fill the SELECT tag using Phalcon\Tag::select. Once the form is submitted, the action “search” is executed in the controller performing the search based on the data entered by the user.

## Performing a Search

The action “search” has a dual behavior. When accessed via POST, it performs a search based on the data sent from the form. But when accessed via GET it moves the current page in the paginator. To differentiate one from another HTTP method, we check it using the *Request* component:

```
<?php

/**
 * Execute the "search" based on the criteria sent from the "index"
 * Returning a paginator for the results
 */
public function searchAction()
{

    if ($this->request->isPost()) {
        //create the query conditions
    } else {
        //paginate using the existing conditions
    }

    //...
}
```

```
}
```

With the help of [Phalcon|Mvc|Model|Criteria](#), we can create the search conditions intelligently based on the data types and values sent from the form:

```
<?php

$query = Criteria::fromInput($this->di, "Products", $_POST);
```

This method verifies which values are different from “” (empty string) and null and takes them into account to create the search criteria:

- If the field data type is text or similar (char, varchar, text, etc.) It uses an SQL “like” operator to filter the results.
- If the data type is not text or similar, it’ll use the operator “=”.

Additionally, “Criteria” ignores all the `$_POST` variables that do not match any field in the table. Values are automatically escaped using “bound parameters”.

Now, we store the produced parameters in the controller’s session bag:

```
<?php

$this->persistent->searchParams = $query->getParams();
```

A session bag, is a special attribute in a controller that persists between requests. When accessed, this attribute injects a [Phalcon|Session|Bag](#) service that is independent in each controller.

Then, based on the built params we perform the query:

```
<?php

$products = Products::find($parameters);
if (count($products) == 0) {
    $this->flash->notice("The search did not found any products");
    return $this->forward("products/index");
}
```

If the search doesn’t return any product, we forward the user to the index action again. Let’s pretend the search returned results, then we create a paginator to navigate easily through them:

```
<?php

$paginator = new Phalcon\Paginator\Adapter\Model(array
    "data" => $products,      //Data to paginate
    "limit" => 5,            //Rows per page
    "page" => $numberPage   //Active page
));

//Get active page in the paginator
$page = $paginator->getPaginate();
```

Finally we pass the returned page to view:

```
<?php

$this->view->setVar("page", $page);
```

In the view (app/views/products/search.phtml), we traverse the results corresponding to the current page:

```
<?php foreach ($page->items as $product) { ?>
<tr>
    <td><?= $product->id ?></td>
    <td><?= $product->getProductTypes()->name ?></td>
    <td><?= $product->name ?></td>
    <td><?= $product->price ?></td>
    <td><?= $product->active ?></td>
    <td><?= $this->tag->linkTo("products/edit/" . $product->id, 'Edit') ?></td>
    <td><?= $this->tag->linkTo("products/delete/" . $product->id, 'Delete') ?></td>
</tr>
<?php } ?>
```

## Creating and Updating Records

Now let's see how the CRUD creates and updates records. From the "new" and "edit" views the data entered by the user are sent to the actions "create" and "save" that perform actions of "creating" and "updating" products respectively.

In the creation case, we recover the data submitted and assign them to a new "products" instance:

```
<?php

/**
 * Creates a product based on the data entered in the "new" action
 */
public function createAction()
{

    $products = new Products();

    $products->id = $this->request->getPost("id", "int");
    $products->product_types_id = $this->request->getPost("product_types_id", "int");
    $products->name = $this->request->getPost("name", "striptags");
    $products->price = $this->request->getPost("price", "double");
    $products->active = $this->request->getPost("active");

    //...
}
```

Data is filtered before being assigned to the object. This filtering is optional, the ORM escapes the input data and performs additional casting according to the column types.

When saving we'll know whether the data conforms to the business rules and validations implemented in the model Products:

```
<?php

/**
 * Creates a product based on the data entered in the "new" action
 */
public function createAction()
{

    //...

    if (!$products->create()) {
```

```

//The store failed, the following messages were produced
foreach ($products->getMessages() as $message) {
    $this->flash->error((string) $message);
}
return $this->forward("products/new");

} else {
    $this->flash->success("Product was created successfully");
    return $this->forward("products/index");
}

}

```

Now, in the case of product updating, first we must present to the user the data that is currently in the edited record:

```

<?php

/**
 * Shows the view to "edit" an existing product
 */
public function editAction($id)
{
    //...

    $product = Products::findFirstById($id);

    $this->tag->setDefault("id", $product->id);
    $this->tag->setDefault("product_types_id", $product->product_types_id);
    $this->tag->setDefault("name", $product->name);
    $this->tag->setDefault("price", $product->price);
    $this->tag->setDefault("active", $product->active);

}

```

The “setDefault” helper sets a default value in the form on the attribute with the same name. Thanks to this, the user can change any value and then sent it back to the database through to the “save” action:

```

<?php

/**
 * Updates a product based on the data entered in the "edit" action
 */
public function saveAction()
{
    //...

    //Find the product to update
    $id = $this->request->getPost("id");
    $product = Products::findFirstById($id);
    if (!$product) {
        $this->flash->error("products does not exist " . $id);
        return $this->forward("products/index");
    }

    //... assign the values to the object and store it

```

```
}
```

### 2.5.11 Changing the Title Dynamically

When you browse between one option and another will see that the title changes dynamically indicating where we are currently working. This is achieved in each controller initializer:

```
<?php

class ProductsController extends ControllerBase
{

    public function initialize()
    {
        //Set the document title
        $this->tag->setTitle('Manage your product types');
        parent::initialize();
    }

    //...
}

}
```

Note, that the method parent::initialize() is also called, it adds more data to the title:

```
<?php

class ControllerBase extends Phalcon\Mvc\Controller
{

    protected function initialize()
    {
        //Prepend the application name to the title
        $this->tag->prependTitle('INVO | ');
    }

    //...
}
```

Finally, the title is printed in the main view (app/views/index.phtml):

```
<!DOCTYPE html>
<html>
    <head>
        <?php echo $this->tag->getTitle() ?>
    </head>
    <!-- ... -->
</html>
```

### 2.5.12 Conclusion

This tutorial covers many more aspects of building applications with Phalcon, hope you have served to learn more and get more out of the framework.

## 2.6 Урок 3: Создание простейшего REST API

В этом уроке мы объясним, как создать простейшее приложение, предоставляющее RESTful API с использованием различных HTTP методов:

- GET для получения и поиска данных
- POST для добавления данных
- PUT для обновления данных
- DELETE для удаления данных

### 2.6.1 Определение API

Наше API содержит следующие методы:

Метод	URL	Действие
GET	/api/robots	Возвращает всех роботов
GET	/api/robots/search/Astro	производит поиск роботов с “Astro” в имени
GET	/api/robots/2	Возвращает робота по его ключу (primary key)
POST	/api/robots	Добавляет нового робота
PUT	/api/robots/2	Обновляет робота по его ключу
DELETE	/api/robots/2	Удаляет робота по его ключу

### 2.6.2 Создание приложения

Поскольку приложение очень простое, мы не будем включать полное MVC окружение для его разработки. В этом случае для достижения нашей цели мы будем использовать *micro application*.

Такой структуры файлов будет более, чем достаточно:

```
my-rest-api/
  models/
    Robots.php
  index.php
  .htaccess
```

Прежде всего, нам понадобится файл .htaccess, который содержит все правила перенаправления URI на файл index.php. Для нашего приложения он будет таким:

```
<IfModule mod_rewrite.c>
  RewriteEngine On
  RewriteCond %{REQUEST_FILENAME} !-f
  RewriteRule ^(.*)$ index.php?url=$1 [QSA,L]
</IfModule>
```

После этого, создаём файл index.php:

```
<?php
$app = new \Phalcon\Mvc\Micro();
// тут определяются роуты
$app->handle();
```

Теперь мы пропишем роуты, как определили выше:

```
<?php

$app = new Phalcon\Mvc\Micro();

// Получение списка всех роботов
$app->get('/api/robots', function() {

});

// Поиск роботов с $name в названии
$app->get('/api/robots/search/{name}', function($name) {

});

// Получение робота по указанному ключу
$app->get('/api/robots/{id:[0-9]+}', function($id) {

});

// Добавление нового робота
$app->post('/api/robots', function() {

});

// Обновление робота по ключу
$app->put('/api/robots/{id:[0-9]+}', function() {

});

// Удаление робота по ключу
$app->delete('/api/robots/{id:[0-9]+}', function() {

});

$app->handle();
```

Каждый роут задан с помощью метода таким же названием, что и HTTP метод. В качестве первого параметра мы передаём шаблон роута, вторым — обработчик, который, в нашем случае является анонимной функцией. Такой роут как '/api/robots/{id:[0-9]+}' однозначно устанавливает, что параметр "id" должен быть числом.

Когда определено соответствие роутов запрашиваемым URI, тогда приложение выполняет соответствующие им обработчики.

### 2.6.3 Создание модели

Наше API предоставляет информацию о “роботах”, хранящуюся в базе данных. Описанная ниже модель позволяет нам получить доступ к таблице объектно-ориентированным путём. Мы реализуем немного бизнес-правил, используя встроенные валидаторы с простейшими проверками. Мы делаем это, чтобы иметь уверенность в том, что сохраняемые данные отвечают требованиям нашего приложения:

```
<?php

use Phalcon\Mvc\Model,
    Phalcon\Mvc\Model\Message,
    Phalcon\Mvc\Model\Validator\InclusionIn,
```

```

Phalcon\Mvc\Model\Validator\Uniqueness;

class Robots extends Model
{

    public function validation()
    {
        // Тип робота должен быть: droid, mechanical или virtual
        $this->validate(new InclusionIn(
            array(
                "field" => "type",
                "domain" => array("droid", "mechanical", "virtual")
            )
        ));

        // Имя робота должно быть уникально
        $this->validate(new Uniqueness(
            array(
                "field" => "name",
                "message" => "The robot name must be unique"
            )
        ));

        // Год не может быть меньше нулевого
        if ($this->year < 0) {
            $this->appendMessage(new Message("The year cannot be less than zero"));
        }

        // Проверяет, были ли получены какие-либо сообщения при валидации
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}

```

Теперь мы должны настроить соединение с базой данных, чтобы использовать его в этой модели

<?php

```

$di = new \Phalcon\DI\FactoryDefault();

// Настройка сервиса базы данных
$di->set('db', function(){
    return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
        "host" => "localhost",
        "username" => "asimov",
        "password" => "zeroth",
        "dbname" => "robotics"
    ));
});

$app = new \Phalcon\Mvc\Micro($di);

```

## 2.6.4 Получение данных

Сначала мы реализуем обработчик, который отвечает на GET-запрос и возвращает всех доступных роботов. Для выполнения этой задачи будем использовать PHQL, который будет возвращать результат выполнения простого запроса в формате JSON:

```
<?php

// Получение всех роботов
$app->get('/api/robots', function() use ($app) {

    $phql = "SELECT * FROM Robots ORDER BY name";
    $robots = $app->modelsManager->executeQuery($phql);

    $data = array();
    foreach( $robots as $robot){
        $data[] = array(
            'id' => $robot->id,
            'name' => $robot->name,
        );
    }

    echo json_encode($data);
});
```

PHQL позволяет нам писать запросы с помощью высокоуровневого, объектно-ориентированного SQL-диалекта, которые внутри него будут переведены в правильные SQL-операторы в зависимости от используемой СУБД. Условие “use” при определении анонимной функции позволяет нам легко передать некоторые переменные из глобальной области видимости в локальную.

Обработчик поиска по названию будет выглядеть следующим образом:

```
<?php

// Поиск роботов, в названии которых содержится $name
$app->get('/api/robots/search/{name}', function($name) use ($app) {

    $phql = "SELECT * FROM Robots WHERE name LIKE :name: ORDER BY name";
    $robots = $app->modelsManager->executeQuery($phql, array(
        'name' => '%' . $name . '%'
    ));

    $data = array();

    foreach( $robots as $robot){
        $data[] = array(
            'id' => $robot->id,
            'name' => $robot->name,
        );
    }

    echo json_encode($data);
});
```

В нашем случае поиск по полю “id” очень похож, кроме того, мы сообщаем, найден робот или нет:

```
<?php
```

```
// Получение робота по ключу
$app->get('/api/robots/{id:[0-9]+}', function($id) use ($app) {
    $phql = "SELECT * FROM Robots WHERE id = :id:";
    $robot = $app->modelsManager->executeQuery($phql, array(
        'id' => $id
    ))->getFirst();

    //Create a response
    $response = new Phalcon\Http\Response();

    if ($robot == false) {
        $response->setJsonContent(array('status' => 'NOT-FOUND'));
    } else {
        $response->setJsonContent(array(
            'status' => 'FOUND',
            'data' => array(
                'id' => $robot->id,
                'name' => $robot->name
            )
        ));
    }
}

return $response;
});
```

## 2.6.5 Вставка данных

Получая данные в виде JSON-строки, вставленной в тело запроса, мы точно так же используем PHQL для вставки:

```
<?php

// Добавление нового робота
$app->post('/api/robots', function() use ($app) {

    $robot = $app->request->getJsonRawBody();

    $phql = "INSERT INTO Robots (name, type, year) VALUES (:name:, :type:, :year:)";

    $status = $app->modelsManager->executeQuery($phql, array(
        'name' => $robot->name,
        'type' => $robot->type,
        'year' => $robot->year
    ));

    // Формируем ответ
    $response = new Phalcon\Http\Response();

    //Проверка, что вставка произведена успешно
    if ($status->success() == true) {

        // Изменение HTML статуса
        $response->setStatusCode(201, "Created");

        $robot->id = $status->getModel()->id;
    }
});
```

```
$response->setJsonContent(array('status' => 'OK', 'data' => $robot));  
  
} else {  
  
    // Изменение HTML статуса  
    $response->setStatusCode(409, "Conflict");  
  
    //Отправляем сообщение об ошибке клиенту  
    $errors = array();  
    foreach ($status->getMessages() as $message) {  
        $errors[] = $message->getMessage();  
    }  
  
    $response->setJsonContent(array('status' => 'ERROR', 'messages' => $errors));  
}  
  
return $response;  
});
```

## 2.6.6 Обновление данных

Обновление данных аналогично их вставке. Полученный параметр “id” сообщает о том, информацию о каком роботе необходимо обновить:

```
<?php  
  
// Обновление робота по ключу  
$app->put('/api/robots/{id:[0-9]+}', function($id) use($app) {  
  
    $robot = $app->request->getJsonRawBody();  
  
    $phql = "UPDATE Robots SET name = :name:, type = :type:, year = :year: WHERE id = :id:";  
    $status = $app->modelsManager->executeQuery($phql, array(  
        'id' => $id,  
        'name' => $robot->name,  
        'type' => $robot->type,  
        'year' => $robot->year  
    ));  
  
    // Формируем ответ  
    $response = new Phalcon\Http\Response();  
  
    // Проверка, что обновление произведено успешно  
    if ($status->success() == true) {  
        $response->setJsonContent(array('status' => 'OK'));  
    } else {  
  
        //Изменение HTML статуса  
        $response->setStatusCode(409, "Conflict");  
  
        $errors = array();  
        foreach ($status->getMessages() as $message) {  
            $errors[] = $message->getMessage();  
        }  
  
        $response->setJsonContent(array('status' => 'ERROR', 'messages' => $errors));  
    }  
});
```

```
        return $response;
});
```

## 2.6.7 Удаление данных

Удаление очень похоже на обновление. Полученный параметр “id” сообщает о том, какого робота необходимо удалить:

```
<?php

// Удаление робота по ключу
$app->delete('/api/robots/{id:[0-9]+}', function($id) use ($app) {

    $phql = "DELETE FROM Robots WHERE id = :id:";
    $status = $app->modelsManager->executeQuery($phql, array(
        'id' => $id
    ));

    // Формируем ответ
    $response = new Phalcon\Http\Response();

    if ($status->success() == true) {
        $response->setJsonContent(array('status' => 'OK'));
    } else {

        // Изменение HTTP статуса
        $response->setStatusCode(409, "Conflict");

        $errors = array();
        foreach ($status->getMessages() as $message) {
            $errors[] = $message->getMessage();
        }

        $response->setJsonContent(array('status' => 'ERROR', 'messages' => $errors));
    }
}

return $response;
});
```

## 2.6.8 Тестирование приложения

Используя `curl` мы протестируем все роуты нашего приложения для проверки правильности его функционирования:

Получение всех роботов:

```
curl -i -X GET http://localhost/my-rest-api/api/robots
```

```
HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 07:05:13 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 117
Content-Type: text/html; charset=UTF-8
```

```
[{"id": "1", "name": "Robotina"}, {"id": "2", "name": "Astro Boy"}, {"id": "3", "name": "Terminator"}]
```

Поиск робота по имени:

```
curl -i -X GET http://localhost/my-rest-api/api/robots/search/Astro
```

```
HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 07:09:23 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 31
Content-Type: text/html; charset=UTF-8
```

```
[{"id": "2", "name": "Astro Boy"}]
```

Получение робота по id:

```
curl -i -X GET http://localhost/my-rest-api/api/robots/3
```

```
HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 07:12:18 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 56
Content-Type: text/html; charset=UTF-8
```

```
{"status": "FOUND", "data": {"id": "3", "name": "Terminator"}}
```

Добавление робота:

```
curl -i -X POST -d '{"name": "C-3PO", "type": "droid", "year": 1977}'
http://localhost/my-rest-api/api/robots
```

```
HTTP/1.1 201 Created
Date: Wed, 12 Sep 2012 07:15:09 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 75
Content-Type: text/html; charset=UTF-8
```

```
{"status": "OK", "data": {"name": "C-3PO", "type": "droid", "year": 1977, "id": "4"}}
```

Попытка добавить робота с уже существующим именем:

```
curl -i -X POST -d '{"name": "C-3PO", "type": "droid", "year": 1977}'
http://localhost/my-rest-api/api/robots
```

```
HTTP/1.1 409 Conflict
Date: Wed, 12 Sep 2012 07:18:28 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 63
Content-Type: text/html; charset=UTF-8
```

```
{"status": "ERROR", "messages": ["The robot name must be unique"]}
```

Или обновление робота с непонятным типом:

```
curl -i -X PUT -d '{"name": "ASIMO", "type": "humanoid", "year": 2000}'
http://localhost/my-rest-api/api/robots/4
```

```
HTTP/1.1 409 Conflict
Date: Wed, 12 Sep 2012 08:48:01 GMT
```

```
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 104
Content-Type: text/html; charset=UTF-8

{"status":"ERROR","messages":["Value of field 'type' must be part of
list: droid, mechanical, virtual"]}
```

И, наконец, удаление робота:

```
curl -i -X DELETE http://localhost/my-rest-api/api/robots/4
```

```
HTTP/1.1 200 OK
Date: Wed, 12 Sep 2012 08:49:29 GMT
Server: Apache/2.2.22 (Unix) DAV/2
Content-Length: 15
Content-Type: text/html; charset=UTF-8

{"status":"OK"}
```

## 2.6.9 Заключение

Как видно, с помощью Phalcon легко разработать RESTful API. Позже мы подробно объясним в документации как использовать микро-приложения и язык *PHQL*.

## 2.7 Список примеров

Следующие примеры законченных приложений, которые вы можете использовать, чтобы узнать больше о Phalcon и использовать их как основу для собственных веб-сайтов/приложений:

## 2.8 Использование Dependency Injection

Следующий пример немного длинный, но объясняет использование контейнера сервисов (service container), service location и dependency injection. Итак, представим, что мы разрабатываем компонент, назовём его *SomeComponent*. Сейчас нам не важно, какую именно задачу он выполняет. Наш компонент имеет некоторую зависимость, отвечающую за соединение с базой данных.

В первом примере соединение устанавливается внутри компонента. Такой подход не является практическим, он не позволяет нам изменить параметры соединения или тип СУБД, потому как компонент работает только так, как был создан.

```
<?php

class SomeComponent
{

    /**
     * Объект соединения жестко вписан в компонент,
     * что усложняет его замену на какой-то
     * внешний или изменение его поведения
     */
    public function someDbTask()
{
```

```
$connection = new Connection(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "secret",
    "dbname" => "invo"
));
// ...
}

$some = new SomeComponent();
$some->someDbTask();
```

Чтобы решить эту проблему, создадим сеттер (setter), который внедрит внешнюю зависимость перед использованием. Теперь это похоже на хорошее решение:

```
<?php

class SomeComponent
{

    protected $_connection;

    /**
     * Назначает внешнее соединение
     */
    public function setConnection($connection)
    {
        $this->_connection = $connection;
    }

    public function someDbTask()
    {
        $connection = $this->_connection;

        // ...
    }
}

$some = new SomeComponent();

// Создание соединения с БД
$connection = new Connection(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "secret",
    "dbname" => "invo"
));

// Внедрение соединения в компонент
$some->setConnection($connection);

$some->someDbTask();
```

Теперь примем во внимание тот факт, что мы используем компонент в различных частях приложения, поэтому появляется необходимость создавать соединение несколько раз и передавать его в компонент. С

помощью некоторого глобального регистра будем получать копию соединения, тем самым нам больше нет необходимости создавать его вновь и вновь:

```
<?php

class Registry
{

    /**
     * Возвращаем соединение
     */
    public static function getConnection()
    {
        return new Connection(array(
            "host" => "localhost",
            "username" => "root",
            "password" => "secret",
            "dbname" => "invo"
        ));
    }

}

class SomeComponent
{

    protected $_connection;

    /**
     * Sets the connection externally
     */
    public function setConnection($connection)
    {
        $this->_connection = $connection;
    }

    public function someDbTask()
    {
        $connection = $this->_connection;

        // ...
    }
}

$some = new SomeComponent();

// Передаем соединение, определяемое в регистре
$some->setConnection(Registry::getConnection());

$some->someDbTask();
```

Теперь представим, что нам необходимо реализовать в компоненте два метода: первый всегда нуждается в создании нового соединения, а второй всегда использует уже установленное (shared):

```
<?php

class Registry
{
```

```
protected static $_connection;

/**
 * Создаёт соединение
 */
protected static function _createConnection()
{
    return new Connection(array(
        "host" => "localhost",
        "username" => "root",
        "password" => "secret",
        "dbname" => "invo"
    ));
}

/**
 * Создаёт соединение единожды и возвращает его
 */
public static function getSharedConnection()
{
    if (self::$_connection==null){
        $connection = self::_createConnection();
        self::$_connection = $connection;
    }
    return self::$_connection;
}

/**
 * Всегда возвращает новое соединение
 */
public static function getNewConnection()
{
    return self::_createConnection();
}

}

class SomeComponent
{

protected $_connection;

/**
 * Назначает внешнее соединение
 */
public function setConnection($connection)
{
    $this->_connection = $connection;
}

/**
 * Для этого метода всегда требуется уже установленное соединение
 */
public function someDbTask()
{
    $connection = $this->_connection;
    // ...
}
```

```

}

/**
 * Для этого метода всегда требуется новое соединение
 */
public function someOtherDbTask($connection)
{
}

}

$some = new SomeComponent();

// Тут внедряется уже установленное (shared) соединение
$some->setConnection(Registry::getSharedConnection());

$some->someDbTask();

// А здесь всегда в качестве параметра передаётся новое соединение
$some->someOtherDbTask(Registry::getNewConnection());

```

До сих пор мы рассматривали случаи, когда внедрение зависимостей решает наши задачи. Передача зависимости в качестве аргументов вместо создания их внутри кода делает наше приложение более гибким и уменьшает его связанность. Однако, в перспективе, такая форма внедрения зависимостей имеет некоторые недостатки.

Например, если компонент имеет много зависимостей, мы будем вынуждены создавать сеттеры с множеством аргументов для передачи зависимостей или конструктор, который принимает их в качестве большого числа аргументов, вдобавок к этому, всякий раз создавать ещё и сами зависимости до использования компонента. Это сделает наш код слишком сложным для сопровождения:

```

<?php

// Создание зависимостей или получение их из регистра
$connection = new Connection();
$session = new Session();
$fileSystem = new FileSystem();
$filter = new Filter();
$selector = new Selector();

// Передача их в конструктор в качестве параметров
$some = new SomeComponent($connection, $session, $fileSystem, $filter, $selector);

// ... или использование сеттеров

$some->setConnection($connection);
$some->setSession($session);
$some->setFileSystem($fileSystem);
$some->setFilter($filter);
$some->setSelector($selector);

```

Думаю, пришлось бы создавать этот объект во многих частях нашего приложения. Если когда-нибудь мы перестанем нуждаться в какой-либо зависимости, нам придётся пройтись по всем этим местам и удалить соответствующий параметр в вызовах конструктора или сеттерах. Чтобы решить эту проблему, вернёмся к глобальному регистру для создания компонента. Однако, это добавит новый уровень абстракции, предшествующий созданию объекта:

```
<?php

class SomeComponent
{
    // ...

    /**
     * Определение метода factory, который создаёт экземпляр SomeComponent и внедряет в него зависимости
     */
    public static function factory()
    {

        $connection = new Connection();
        $session = new Session();
        $fileSystem = new FileSystem();
        $filter = new Filter();
        $selector = new Selector();

        return new self($connection, $session, $fileSystem, $filter, $selector);
    }
}
```

Минуточку, мы снова вернулись туда, откуда начали: создание зависимостей внутри компонента! Мы можем двигаться дальше и находить способ решать эту проблему каждый раз. Но, это означает, что мы снова и снова будем наступать на те же грабли.

Практически применимый и элегантный способ решить эту проблему — это использовать контейнер для зависимостей. Он играет ту же роль, что и глобальный регистр, который мы видели выше. Использование контейнера в качестве моста к зависимостям позволяет нам уменьшить сложность нашего компонента:

```
<?php

class SomeComponent
{

    protected $_di;

    public function __construct($di)
    {
        $this->_di = $di;
    }

    public function someDbTask()
    {

        // Получение сервиса соединений
        // Всегда возвращает соединение
        $connection = $this->_di->get('db');

    }

    public function someOtherDbTask()
    {

        // Получение сервиса соединения, предназначенного для общего доступа,
    }
}
```

```

// всегда возвращает одно и то же соединение
$connection = $this->_di->getShared('db');

// Этот метод так же требует сервиса фильтрации входных данных
$filter = $this->_di->get('filter');

}

}

$di = new Phalcon\DI();

// Регистрация в контейнере сервиса "db"
$di->set('db', function() {
    return new Connection(array(
        "host" => "localhost",
        "username" => "root",
        "password" => "secret",
        "dbname" => "invo"
    ));
});

// Регистрация в контейнере сервиса "filter"
$di->set('filter', function() {
    return new Filter();
});

// Регистрация в контейнере сервиса "session"
$di->set('session', function() {
    return new Session();
});

// Передача контейнера сервисов в качестве единственного параметра
$some = new SomeComponent($di);

$some->someTask();

```

Теперь компонент имеет простой доступ к сервисам, которые ему необходимы. Если сервис не востребован, он не будет инициализирован, тем самым экономя ресурсы. Так же компонент теперь обладает низкой связанностью. Например, можно заменить способ создания соединений, поведение или любой другой аспект их работы, и это никак не отразится на компоненте.

### 2.8.1 Наш подход

Phalcon\DI — это компонент, реализующий Dependency Injection и Location сервисов и является контейнером для них.

Поскольку Phalcon обладает низкой связанностью, Phalcon\DI необходимо обеспечить интеграцию различных компонентов фреймворка. Разработчики так же могут использовать этот компонент для внедрения зависимостей и использования глобальных экземпляров различных классов, используемых в приложении.

В основе своей, компонент реализует паттерн [Инверсии управления](#). Применяя его, объекты получают их зависимости не с использованием сеттеров или конструкторов, а с помощью сервиса внедрения зависимостей. Это снижает общую сложность, поскольку остаётся только один способ получения зависимостей в компоненте.

К тому же, этот паттерн увеличивает тестируемость в коде, что позволяет снизить “ошибочность” кода.

## 2.8.2 Регистрация сервисов в Контейнере сервисов

Регистрация сервисов возможна как разработчиком, так и самим фреймворком. Когда компоненту А требуется компонент В (или экземпляр его класса) для работы, он может запросить его из контейнера, а не создавать новый экземпляра.

Такой способ работы даёт нам много преимуществ:

- Мы можем легко заменять компонент на созданный нами или кем-то другим.
- Мы обладаем полным контролем над инициализацией объекта, что позволяет нам настраивать эти объекты так, как нам необходимо, прежде, чем передать их компонентам.
- Мы можем получать глобальный экземпляр компонента структурированным и унифицированным образом.

Зарегистрировать сервисы можно несколькими различными способами:

```
<?php

// Создание контейнера Dependency Injector
$di = new Phalcon\DI();

// По названию класса
$di->set("request", 'Phalcon\Http\Request');

// С использованием анонимной функции для отложенной загрузки
$di->set("request", function() {
    return new Phalcon\Http\Request();
});

// Регистрация экземпляра напрямую
$di->set("request", new Phalcon\Http\Request());

// Определение с помощью массива
$di->set("request", array(
    "className" => 'Phalcon\Http\Request'
));
```

Для регистрации сервисов можно так же использовать синтаксис массивов:

```
<?php

// Создание контейнера DI
$di = new Phalcon\DI();

// По названию класса
$di["request"] = 'Phalcon\Http\Request';

// С использованием анонимной функции для отложенной загрузки
$di["request"] = function() {
    return new Phalcon\Http\Request();
};

// Регистрация экземпляра напрямую
$di["request"] = new Phalcon\Http\Request();
```

```
// Определение с помощью массива
$di["request"] = array(
    "className" => 'Phalcon\Http\Request'
);
```

В примере, данном выше, когда фреймворк нуждается в доступе к запрашиваемым данным, он будет запрашивать в контейнере сервис, названный ‘request’. Контейнер, в свою очередь, возвращает экземпляр затребованного сервиса. Разработчик, в конечном итоге, может заменить компонент, когда захочет.

Каждый из методов регистрации сервисов имеет свои достоинства и недостатки. Какой из них использовать — зависит только от разработчика и от конкретных требований.

Назначение сервиса строкой очень простое, но лишено гибкости. В качестве массива — предоставляет большую гибкость, но делает код менее понятным. Анонимные функции неплохо балансируют между этими двумя способами, но им может потребоваться больше обслуживания, чем это ожидается.

Phalcon\DI предоставляет отложенную загрузку для каждого хранимого им сервиса. Если разработчик не решит создавать экземпляр объекта напрямую и хранить его в контейнере, любой объект сохранённый в нём (через массив, строку и т.д.) будет загружен отложенно (lazy load), т.е. создастся только тогда, когда будет востребован.

## Простая регистрация

Как было показано выше, есть несколько способов для регистрации сервисов. Следующие из них мы называем “простыми”:

### Строчный

Этот способ ожидает в качестве параметра имя существующего класса, возвращает его объект, если класс не был загружен автoloадером. Такой способ не позволяет передавать аргументы для конструктора класса или настраивать параметры:

```
<?php
```

```
// Возвращает новый Phalcon\Http\Request();
$di->set('request', 'Phalcon\Http\Request');
```

### Объект

Этот способ в качестве параметра принимает объект. Объект не нуждается в создании, потому как объект уже является объектом сам по себе. Вообще говоря, в данном случае это не является настоящим внедрением зависимости, однако такой способ вполне используем, если вы хотите быть уверены в том, что возвращаемая зависимость всегда будет одним и тем же объектом/значением:

```
<?php
```

```
// Возвращает новый Phalcon\Http\Request();
$di->set('request', new Phalcon\Http\Request());
```

## Замыкания/Анонимные функции

Этот метод дает больше свободы для построения зависимости, если этого захочет, тем не менее, он весьма сложен в плане изменения некоторых параметров извне без полного замещения определения зависимости:

```
<?php

$di->set("db", function() {
    return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
        "host" => "localhost",
        "username" => "root",
        "password" => "secret",
        "dbname" => "blog"
    ));
});
```

Некоторые ограничения можно преодолеть путём передачи дополнительных переменных в область видимости замыкания:

```
<?php

//Using the $config variable in the current scope
$di->set("db", function() use ($config) {
    return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
        "host" => $config->host,
        "username" => $config->username,
        "password" => $config->password,
        "dbname" => $config->name
    ));
});
```

## Сложная регистрация

Если потребуется изменить определение сервиса без создания экземпляра, тогда нам придётся определять его с использованием синтаксиса массивов. Такое определение может оказаться чуть более длинным:

```
<?php

// Регистрация сервиса 'logger' с помощью имени класса и параметров для него
$di->set('logger', array(
    'className' => 'Phalcon\Logger\Adapter\File',
    'arguments' => array(
        array(
            'type' => 'parameter',
            'value' => '../apps/logs/error.log'
        )
    )
));

// Или в виде анонимной функции
$di->set('logger', function() {
    return new \Phalcon\Logger\Adapter\File('../apps/logs/error.log');
});
```

Оба способа приведут к одинаковому результату. Определение же с помощью массива позволяет изменение параметров, если это необходимо:

```
<?php

// Изменение названия класса для сервиса
$di->getService('logger')->setClassName('MyCustomLogger');

// Изменение первого параметра без пересоздания экземпляра сервиса logger
$di->getService('logger')->setParameter(0, array(
    'type' => 'parameter',
    'value' => '../apps/logs/error.log'
));
```

В дополнение к этому, используя синтаксис массивов, можно использовать три типа внедрения зависимостей:

### Constructor Injection

Этот тип передаёт зависимости/аргументы в конструктор класса. Представим, что у нас есть следующий компонент:

```
<?php

namespace SomeApp;

use Phalcon\Http\Response;

class SomeComponent
{

    protected $_response;

    protected $_someFlag;

    public function __construct(Response $response, $someFlag)
    {
        $this->_response = $response;
        $this->_someFlag = $someFlag;
    }
}
```

Сервис может быть зарегистрирован следующим образом:

```
<?php

$di->set('response', array(
    'className' => 'Phalcon\Http\Response'
));

$di->set('someComponent', array(
    'className' => 'SomeApp\SomeComponent',
    'arguments' => array(
        array('type' => 'service', 'name' => 'response'),
        array('type' => 'parameter', 'value' => true)
    )
));
```

Сервис “response” (Phalcon\Http\Response) передаётся в конструктор в качестве первого параметра, в то время как вторым параметром передаётся булевое значение (true) без изменений.

### Setter Injection

Классы могут иметь сеттеры для внедрения дополнительных зависимостей. Наш предыдущий класс может быть изменён, чтобы принимать зависимости с помощью сеттеров:

```
<?php

namespace SomeApp;

use Phalcon\Http\Response;

class SomeComponent
{

    protected $_response;

    protected $_someFlag;

    public function setResponse(Response $response)
    {
        $this->_response = $response;
    }

    public function setFlag($someFlag)
    {
        $this->_someFlag = $someFlag;
    }

}
```

Сервис с сеттерами для зависимостей может быть зарегистрирован следующим образом:

```
<?php

$di->set('response', array(
    'className' => 'Phalcon\Http\Response'
));

$di->set('someComponent', array(
    'className' => 'SomeApp\SomeComponent',
    'calls' => array(
        array(
            'method' => 'setResponse',
            'arguments' => array(
                array('type' => 'service', 'name' => 'response'),
            )
        ),
        array(
            'method' => 'setFlag',
            'arguments' => array(
                array('type' => 'parameter', 'value' => true)
            )
        ),
    )
));
});
```

## Properties Injection

Менее распространённым способом является внедрение зависимостей или полей класса напрямую:

```
<?php

namespace SomeApp;

use Phalcon\Http\Response;

class SomeComponent
{

    public $response;

    public $someFlag;

}
```

Сервис с прямым внедрением может быть зарегистрирован следующим способом:

```
<?php

$di->set('response', array(
    'className' => 'Phalcon\Http\Response',
));

$di->set('someComponent', array(
    'className' => 'SomeApp\SomeComponent',
    'properties' => array(
        array(
            'name' => 'response',
            'value' => array('type' => 'service', 'name' => 'response')
        ),
        array(
            'name' => 'someFlag',
            'value' => array('type' => 'parameter', 'value' => true)
        )
    )
));
```

Поддерживаются параметры следующих типов:

| Тип       | Описание  | Пример  |
|-----------|---|---|
| parameter | Буквенное значение, передаваемое в качестве параметра | array('type' => 'parameter', 'value' => 1234)                                       |
| service   | Другой сервис в контейнере                            | array('type' => 'service', 'name' => 'request')                                     |
| instance  | Объект, который должен создаваться динамически        | array('type' => 'instance', 'className' => 'DateTime', 'arguments' => array('now')) |

Получение сервисов, определение которых весьма сложно может быть немного медленнее, чем рассмотренные выше определения. Однако, это предоставляет больше возможностей для определения и внедрения сервисов.

Можно совмещать различные типы определения, определяя для себя наиболее подходящий способ регистрации сервиса в соответствии с потребностями приложения.

## 2.8.3 Доступ к сервисам

Получение сервиса из контейнера очень просто производится вызовом метода “get”. Будет возвращен новый экземпляр сервиса:

```
<?php $request = $di->get("request");
```

Так же можно вызвать магический метод:

```
<?php
```

```
$request = $di->getRequest();
```

Или использовать доступ как к массиву:

```
<?php
```

```
$request = $di['request'];
```

Аргументы могут быть переданы в конструктор добавлением массива параметров в метод “get”:

```
<?php
```

```
// новый MyComponent("some-parameter", "other")
$component = $di->get("MyComponent", array("some-parameter", "other"));
```

## 2.8.4 Совместный доступ к сервисам

Сервисы могут быть сразу зарегистрированы, как предназначенные для совместного (“shared”) доступа. Это означает, что они всегда будут синглетонами ([singletons](#)). После того, как этот сервис будет один раз создан, всегда будет возвращаться тот же самый его экземпляр:

```
<?php
```

```
// Регистрация сервиса сессий, как "always shared"
$di->setShared('session', function() {
    $session = new Phalcon\Session\Adapter\Files();
    $session->start();
    return $session;
});

$session = $di->get('session'); // Locates the service for the first time
$session = $di->getSession(); // Returns the first instantiated object
```

Так же можно зарегистрировать сервис с совместным доступом, передав “true” в качестве третьего параметра метода “set”:

```
<?php
```

```
// Регистрация сервиса сессий, как "always shared"
$di->set('session', function() {
    //...
}, true);
```

Если сервис не был зарегистрирован для общего доступа и вы хотите всё же получать один и тот же экземпляр каждый раз, то можно получать его, используя метод DI “getShared”:

```
<?php

$request = $di->getShared("request");
```

## 2.8.5 Ручное управление сервисами

После того, как сервис был зарегистрирован в контейнере, вы можете управлять им вручную:

```
<?php

// Регистрация сервиса сессий
$di->set('request', 'Phalcon\Http\Request');

// Получение сервиса
$requestService = $di->getService('request');

// Изменение его определение
$requestService->setDefinition(function() {
    return new Phalcon\Http\Request();
});

// Назначение его как "always shared"
$requestService->setShared(true);

//Получение сервиса (возвращает экземпляр Phalcon\Http\Request)
$request = $requestService->resolve();
```

## 2.8.6 Создание экземпляров классов через контейнер сервисов

Когда вы запрашиваете какой-то сервис из контейнера, и он не может найти его по такому имени, контейнер пытается загрузить класс с таким же названием. С помощью этого вы можете легко заменить какой-либо класс на любой другой, зарегистрировав сервис с таким же названием:

```
<?php

// Регистрация контроллера как сервиса
$di->set('IndexController', function() {
    $component = new Component();
    return $component;
}, true);

// Регистрация компонента как сервиса
$di->set('MyOtherComponent', function() {
    //Actually returns another component
    $component = new AnotherComponent();
    return $component;
});

// Создание экземпляра объекта с помощью контейнера сервисов
$myComponent = $di->get('MyOtherComponent');
```

Вы можете пользоваться этим, всегда создавая экземпляры объектов ваших классов с помощью контейнера сервисов (даже если они не регистрировались как сервисы). DI будет запускать правильный автозагрузчик для того, чтобы в итоге загрузить класс. Делая так, вы сможете легко заменить любой класс в будущем, реализовав его определение.

## 2.8.7 Автоматическое внедрение DI

Если класс или компонент требует DI для нахождения сервисов, DI может автоматически внедрить себя в экземпляры этих компонентов или объектов, чтобы сделать это вам необходимо реализовать `Phalcon\DI\InjectionAwareInterface` в своём классе:

```
<?php

class MyClass implements \Phalcon\DI\InjectionAwareInterface
{

    protected $_di;

    public function setDi($di)
    {
        $this->_di = $di;
    }

    public function getDi()
    {
        return $this->_di;
    }

}
```

Когда сервис будет запрошен, `$di` будет передан в `setDi` автоматически:

```
<?php

// Регистрация сервиса
$di->set('myClass', 'MyClass');

// Получение сервиса (ВНИМАНИЕ: $myClass->setDi($di) вызовется автоматически)
$myClass = $di->get('myClass');
```

## 2.8.8 Избежание разрешения сервисов

Сервисы, которые используются при каждом обращении к приложению, могут избежать процесса их разрешения, что может немного увеличить производительность:

```
<?php

// Внешнее разрешение объекта вместо его определения
$route = new MyRouter();

// Передача уже созданного объекта
$di->set('router', $route);
```

## 2.8.9 Размещение сервисов в файлах

Вы можете улучшить организацию вашего приложения переместив регистрацию сервисов в отдельные файлы, которые делают всё, что происходит при старте приложения:

```
<?php

$di->set('router', function() {
```

```
    return include "../app/config/routes.php";
};


```

А файл ”../app/config/routes.php” вернёт готовый объект:

```
<?php

$router = new MyRouter();

$router->post('/login');

return $router;
```

### 2.8.10 Статический доступ к DI

При необходимости вы можете получить доступ к последнему созданному DI в статической функции следующим образом:

```
<?php

class SomeComponent
{

    public static function someMethod()
    {
        // Получение сервиса сессий
        $session = Phalcon\DI::getDefault()->getSession();
    }
}
```

### 2.8.11 Factory Default DI

Несмотря на то, что разрозненный характер Phalcon дарит нам огромную свободу и гибкость, возможно мы захотим легко использовать полноценный фреймворк. Для достижения этой цели фреймворк предоставляет Phalcon\DI называющийся Phalcon\DI\FactoryDefault. Этот класс автоматически регистрирует такие сервисы, которые обычно определены в полноценном фреймворке.

```
<?php $di = new Phalcon\DI\FactoryDefault();
```

### 2.8.12 Соглашение именования сервисов

Хотя, вы и можете регистрировать сервисы с любыми именами, какие вам только понравятся, Phalcon имеет некоторое соглашение именования сервисов, что позволяет ему правильно работать с сервисами, когда они вам необходимы.

| Название сервиса   | Описание                                | По умолчанию  | Общий доступ |
|--------------------|---|---|--------------|
| dispatcher         | Диспетчер контроллеров                  | <a href="#">Phalcon\ Mvc\ Dispatcher</a>                | Да           |
| router             | Роутер                                  | <a href="#">Phalcon\ Mvc\ Router</a>                    | Да           |
| url                | Генератор URL'ов                        | <a href="#">Phalcon\ Mvc\ Url</a>                       | Да           |
| request            | Окружение HTTP запросов                 | <a href="#">Phalcon\ Http\ Request</a>                  | Да           |
| response           | Окружение HTTP ответов                  | <a href="#">Phalcon\ Http\ Response</a>                 | Да           |
| cookies            | Сервис управления HTTP Cookies          | <a href="#">Phalcon\ Http\ Response\ Cookies</a>        | Да           |
| filter             | Входной фильтр                          | <a href="#">Phalcon\ Filter</a>                         | Да           |
| flash              | Всплывающие сообщения                   | <a href="#">Phalcon\ Flash\ Direct</a>                  | Да           |
| flashSession       | Сессия всплывающих сообщений            | <a href="#">Phalcon\ Flash\ Session</a>                 | Да           |
| session            | Сессия                                  | <a href="#">Phalcon\ Session\ Adapter</a>               | Да           |
| eventsManager      | Управление событиями                    | <a href="#">Phalcon\ Events\ Manager</a>                | Да           |
| db                 | Низкоуровневый коннектор к базе данных  | <a href="#">Phalcon\ Db</a>                             | Да           |
| security           | Помощник безопасности                   | <a href="#">Phalcon\ Security</a>                       | Да           |
| crypt              | Encrypt/Decrypt data                    | <a href="#">Phalcon\ Crypt</a>                          | Да           |
| tag                | генератор HTML конструкций              | <a href="#">Phalcon\ Tag</a>                            | Да           |
| escaper            | Контекстное экранирование               | <a href="#">Phalcon\ Escaper</a>                        | Да           |
| annotations        | Парсер аннотаций                        | <a href="#">Phalcon\ Annotations\ Adapter</a>           | Да           |
| modelsManager      | Управление моделями                     | <a href="#">Phalcon\ Mvc\ Model\ Manager</a>            | Да           |
| modelsMetadata     | Мета-данные моделей                     | <a href="#">Phalcon\ Mvc\ Model\ MetaData</a>           | Да           |
| transactionManager | Управление транзакциями моделей         | <a href="#">Phalcon\ Mvc\ Model\ TransactionManager</a> | Да           |
| modelsCache        | Кэширование для моделей                 | None  | •            |
| viewsCache         | Кэширование для частичных представлений | None  | •            |

### 2.8.13 Реализация собственного DI

Для создания собственного DI необходимо реализовать интерфейс [Phalcon\ DiInterface](#), или использовать наследование и переопределить стандартный компонент Phalcon.

## 2.9 Архитектура MVC

Phalcon поддерживает использование парадигмы объектно-ориентированного программирования и поддержку классов необходимых для разделения на Model, View, Controller ( кратко [MVC](#)). Этот шаблон проектирования активно используется в других web и desktop платформах.

Преимущества MVC:

- \* Отделение бизнес-логики от пользовательского интерфейса и работы с базой данных
- \* Позволяет располагать разные части в разных местах, что благоприятно оказывается на поддержке и обслуживании

Если вы выберите MVC, все запросы будут выполняться по MVC архитектуре. Phalcon поддерживает MVC своими классами написанными на C, что позволяет работать с повышенной производительностью.

### 2.9.1 Модель ( Models )

Модель отвечает за информацию (данные) приложения и правила управления этими данными. Модели в основном используется для управления соответствующей таблицей базы данных и правил взаимодействия с ней. В большинстве случаев, каждая таблица в базе данных будет соответствовать одной модели в вашем приложении. Основная часть бизнес-логики приложения будет сосредоточена в моделях. [Подробнее про модель](#)

### 2.9.2 Представление ( Views )

Представление отвечает за пользовательский интерфейс вашего приложения. Чаще всего это HTML файлы с вставками PHP кода исключительно вывода данных. Этот слой отвечает за вывод данных в веб-браузер или другой инструмент который обращается к вашему приложению. [Подробнее про представление](#)

### 2.9.3 Контроллер ( Controllers )

Контроллеры обеспечивают обмен данными между моделью и представлением. Контроллеры отвечают за обработку входящих запросов от веб-браузера, запрашивают данные у модели и передают эти данные в представление для вывода. [Подробнее про контроллер](#)

## 2.10 Использование контроллеров

Контроллеры содержат в себе ряд методов, называемых действиями (actions). Действия контроллеров занимаются непосредственно обработкой запросов. По умолчанию все публичные методы контроллеров доступны для доступа по URL. Действия отвечают за разбор запросов (request) и создание ответов (response). Как правило, результаты работы действий используются для представлений, но так же возможно их иное использование.

Например, при обращении по ссылке: <http://localhost/blog/posts/show/2012/the-post-title> Phalcon разберёт её и получит следующие части:

|                              |                |
|------------------------------|----------------|
| <b>Подкаталог Phalcon</b>    | blog           |
| <b>Контроллер/Controller</b> | posts          |
| <b>Действие/Action</b>       | show           |
| <b>Параметр</b>              | 2012           |
| <b>Параметр</b>              | the-post-title |

Для этого случая запрос будет отправлен для обработки в контроллер PostsController. Для контроллеров не существует специального места, в приложениях они загружаются с помощью [автозагрузки](#), поэтому вы можете организовывать их в любую удобную структуру.

Контроллеры должны иметь суффикс “Controller”, а действия, соответственно, “Action”. Образец контроллера:

```
<?php
```

```
class PostsController extends \Phalcon\Mvc\Controller
{
```

```
public function indexAction()
{
}

public function showAction($year, $postTitle)
{
}

}
```

Параметры, определённые в ссылках (URI) передаются в качестве параметров действий, и легко могут быть доступны для локального использования. Контроллеры могут наследоваться от *Phalcon\ Mvc\ Controller*, в таком случае в них можно использовать лёгкий способ получения сервисов приложения.

Параметры без значений по умолчанию обрабатываются по мере необходимости. Установка дополнительных значений производится по PHP стандартам:

```
<?php

class PostsController extends \Phalcon\ Mvc\ Controller
{

    public function indexAction()
    {

    }

    public function showAction($year=2012, $postTitle='другой заголовок по умолчанию')
    {

    }

}
```

Параметры поступают в порядке указанном в правиле маршрутизации (route). Получить доступ к произвольному параметру можно следующим образом:

```
<?php

class PostsController extends \Phalcon\ Mvc\ Controller
{

    public function indexAction()
    {

    }

    public function showAction()
    {
        $year = $this-> dispatcher->getParam('year');
        $postTitle = $this-> dispatcher->getParam('postTitle');
    }

}
```

## 2.10.1 Цикл работы

Цикл работы диспетчера выполняться пока не будет явно остановлен. В примере выше выполняется лишь одно действие. Пример ниже показывает как с использованием метода “forward” можно обеспечить более сложный процесс диспетчеризации, путём перенаправления потока на другой контроллер/действие.

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function showAction($year, $postTitle)
    {
        $this->flash->error("У вас недостаточно прав для выполнения этого действия");

        // Перенаправление на другое действие
        $this->dispatcher->forward(array(
            "controller" => "users",
            "action" => "signin"
        ));
    }
}
```

Если у пользователя недостаточно прав, он будет перенаправлен в контроллер пользователей для выполнения авторизации.

```
<?php

class UsersController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function signinAction()
    {
    }
}
```

Метод “forwards” может быть вызван неограниченное количество раз, приложение будет выполняться, пока не появится явный сигнал для завершения. Если действия, которые должны быть выполнены, в цикле диспетчера завершены, то диспетчер автоматически вызовет MVC слой отображения (View), управляемый компонентом *Phalcon|Mvc|View*.

## 2.10.2 Инициализация контроллеров

Сервис *Phalcon\ Mvc\ Controller* предполагает наличие метода инициализации “initialize”, автоматически выполняемого первым, перед любым другим действием контроллера. Использование метода “`__construct`” не рекомендуется.

```
<?php
```

```
class PostsController extends \Phalcon\ Mvc\ Controller
{
    public $settings;

    public function initialize()
    {
        $this->settings = array(
            "mySetting" => "value"
        );
    }

    public function saveAction()
    {
        if ($this->settings["mySetting"] == "value") {
            //...
        }
    }
}
```

Метод ‘initialize’ вызывается только в том случае, если событие ‘beforeExecuteRoute’ было успешно выполнено. Это позволяет избежать выполнения логики приложения без авторизации.

Если вы хотите выполнить инициализацию логики сразу после создания объекта контроллера, то можно реализовать метод ‘onConstruct’:

```
<?php
```

```
class PostsController extends \Phalcon\ Mvc\ Controller
{
    public function onConstruct()
    {
        //...
    }
}
```

Знайте, что метод ‘onConstruct’ выполняется, даже если Action, который будет выполняться, не существует в контроллере или пользователь не имеет к нему доступа (контроль доступа обеспечивает разработчик).

## 2.10.3 Внедрение сервисов / Injecting Services

Если контроллер использует наследование от *Phalcon\ Mvc\ Controller*, то в нём можно использовать лёгкий способ получения любых сервисов приложения. Например, в приложении имеется зарегистрированный сервис с именем “storage”:

```
<?php

$di = new Phalcon\DI();

$di->set('storage', function() {
    return new Storage('/some/directory');
}, true);
```

Доступ к зарегистрированным сервисам можно получить несколькими способами:

```
<?php

class FilesController extends \Phalcon\Mvc\Controller
{

    public function saveAction()
    {

        // Прямой доступ по имени, используя его как свойство
        $this->storage->save('/some/file');

        // С использованием сервиса DI
        $this->di->get('storage')->save('/some/file');

        // Используя магический метод
        $this->di->getStorage()->save('/some/file');

        // Еще больше магических методов для получения всей цепочки
        $this->getDi()->getStorage()->save('/some/file');

        // Используя синтаксис работы с массивами
        $this->di['storage']->save('/some/file');
    }
}
```

Если вы используете все возможности Phalcon, прочитайте о провайдере сервисов [используемый по умолчанию](#).

## 2.10.4 Запросы Request и ответы Response

Давайте предположим, что фреймворк состоит из набора предварительно зарегистрированных сервисов. В этом примере будет показано как работать с параметрами HTTP. Сервис работы с запросами “request” содержит экземпляр `Phalcon\Http\Request`, а “response” - экземпляр `Phalcon\Http\Response` и обеспечивает отправку собранных данных клиенту.

```
<?php

class PostsController extends Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function saveAction()
```

```
{  
    // Проверка что данные пришли методом POST  
    if ($this->request->isPost() == true) {  
        // Доступ к POST данным  
        $customerName = $this->request->getPost("name");  
        $customerBorn = $this->request->getPost("born");  
    }  
}  
}
```

Объект работы с ответами, как правило, не используется напрямую, но создаваясь до выполнения всех действий может быть полезен, например, в событии afterDispatch - для указания правильного кода ответа:

```
<?php  
  
class PostsController extends Phalcon\Mvc\Controller  
{  
  
    public function indexAction()  
    {  
  
    }  
  
    public function notFoundAction()  
    {  
        // Отправка 404 HTTP статуса  
        $this->response->setStatusCode(404, "Not Found");  
    }  
  
}
```

Получить подробности про работу с HTTP можно в соответствующих статьях *request* и *response*.

## 2.10.5 Данные сессий

Сессии позволяют сохранять данные между запросами. Вы можете использовать доступ к *Phalcon\Session\Bag* из любого контроллера для хранения (encapsulate) постоянных данных.

```
<?php  
  
class UserController extends Phalcon\Mvc\Controller  
{  
  
    public function indexAction()  
    {  
        $this->persistent->name = "Коляня";  
    }  
  
    public function welcomeAction()  
    {  
        echo "Привет, ", $this->persistent->name;  
    }  
  
}
```

## 2.10.6 Использование сервисов как контроллеров

Сервисы могут работать в качестве контроллеров, классы контроллеров первым делом запрашиваются у сервиса контейнеров. Соответственно любой класс, зарегистрированный под именем контроллера, легко может его заменить:

```
<?php

// Регистрация контроллера как сервиса
$di->set('IndexController', function() {
    $component = new Component();
    return $component;
});

// Регистрация контроллера из пространства имен в качестве сервиса
$di->set('Backend\Controllers\IndexController', function() {
    $component = new Component();
    return $component;
});
```

## 2.10.7 Создание базового контроллера (Base Controller)

Некоторые функции в приложении, такие как контроль доступа, перевод, кэширование или шаблонизация, чаще всего однообразны для всех контроллеров приложения. В таких случаях рекомендуется использование “базового контроллера”, что обеспечит поддержку кодом логики DRY (не повторяя). Базовый контроллер, это просто класс расширяющий [Phalcon Mvc Controller](#) и инкапсулирующий общую функциональность, которую должны иметь все контроллеры. Ваши контроллеры, для получения доступа к этой функциональности, должны использовать наследование от базового контроллера.

Этот класс может располагаться где угодно, но для поддержания общей практики, мы рекомендуем располагать его в каталоге контроллеров, например, apps/controllers/ControllerBase.php. Файл может быть напрямую подключен в файле загрузки приложения, но может также загружаться используя автозагрузку.

```
<?php

require "../app/controllers/ControllerBase.php";
```

Реализация общих компонентов (действий, методов, свойств и т.д.) находится в этом файле:

```
<?php

class ControllerBase extends \Phalcon\Mvc\Controller
{

    /**
     * Это действие доступно для всех контроллеров
     */
    public function someAction()
    {

    }
}
```

Теперь любой контроллер, наследуемый от базового, автоматически получает доступ к общим компонентам (смотрите выше):

```
<?php

class UsersController extends ControllerBase
{



}
```

## 2.10.8 События контроллеров

Контролеры автоматически выступают в роли слушателей (listeners) событий *диспетчера*, внедрение методов с определёнными названиями позволяет выполнять события до и после выполнения основных действий.

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function beforeExecuteRoute($dispatcher)
    {
        // Выполняется до запуска любого найденного действия
        if ($dispatcher->getActionName() == 'save') {

            $this->flash->error("You don't have permission to save posts");

            $this->dispatcher->forward(array(
                'controller' => 'home',
                'action' => 'index'
            ));

            return false;
        }
    }

    public function afterExecuteRoute($dispatcher)
    {
        // Выполняется после каждого выполненного действия
    }
}
```

## 2.11 Работа с Моделями

Модель представляет собой информацию (данные) приложения и правила для манипуляции этими данными. Модели в основном используются для управления правилами взаимодействия с соответствующими таблицами базы данных. В большинстве случаев, каждая таблица в вашей базе данных соответствует одной модели в вашем приложении. Большая часть всей бизнес-логики вашего приложения будет сосредоточена в моделях.

*Phalcon\ Mvc\ Model* является родительским классом для всех моделей в вашем приложении Phalcon. Он обеспечивает независимость данных от вашей базы, основные CRUD операции, расширенные поисковые возможности, а также возможность построения зависимостей между моделями. *Phalcon\ Mvc\ Model* исключает необходимость использования SQL запросов, потому как данный класс динамически переводит методы на соответствующие им операции СУБД.

Модели предназначены для работы с базой данных на высшем уровне абстракции. Если вы испытываете потребность в работе с базой данных на низшем уровне, обратитесь к документации компонента [Phalcon\Db](#).

### 2.11.1 Создание Модели

Модель это класс, который расширяет [Phalcon\Mvc\Model](#). Файл с моделью должен быть помещен в директорию `models`. Файл должен содержать только один класс; его имя должно быть записано в CamelCase стиле.

```
<?php

class Robots extends \Phalcon\Mvc\Model
{
```

Пример выше демонстрирует реализацию модели “Robots”. Обратите внимание, что класс `Robots` наследуется от [Phalcon\Mvc\Model](#). Данный компонент предоставляет большой набор функционала для модели, которая наследует его, включая основные операции CRUD (Create, Read, Update, Delete), валидацию данных, а также поддержку усложненного поиска и возможность связывать несколько моделей друг с другом.

Если вы используете PHP 5.4/5.5 рекомендовано объявлять каждый столбец базы данных, который входит в модель в целях экономии памяти и уменьшения общего выделения памяти на выполнение.

По умолчанию модель “Robots” будет ссылаться на таблицу ‘`robots`’. Если вы захотите вручную указать другое имя для маппинга таблицы, вы можете использовать метод `getSource()`:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function getSource()
    {
        return "the_robots";
    }
}
```

Теперь модель `Robots` маппирует (использует) таблицу “`the_robots`”. Метод `initialize()` помогает в создании модели с пользовательским поведением, т.е. использованиии другой таблицы. Метод `initialize()` вызывает лишь однажды во время запроса.

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->setSource("the_robots");
    }
}
```

Метод initialize() вызывается один раз при обработке запроса к приложению и предназначен для инициализации экземпляров модели в приложении. Если вам необходимо произвести некоторые настройки экземпляра объекта после того, как он создан, вы можете использовать метод ‘onConstruct’:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function onConstruct()
    {
        //...
    }

}
```

## Public свойства и Setters/Getters

Модели могут быть реализованы с помощью свойств с общим доступом (public), при этом свойства модели доступны для чтения/изменения из любой части кода без ограничений:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{
    public $id;

    public $name;

    public $price;
}
```

При использовании getters и setters вы можете полностью контролировать видимость свойств, их обработку и, например, применять различную валидацию при сохранении объекта:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{
    protected $id;

    protected $name;

    protected $price;

    public function getId()
    {
        return $this->id;
    }

    public function setName($name)
    {
        if (strlen($name) < 10) {
            throw new \InvalidArgumentException('Имя слишком короткое');
        }
        $this->name = $name;
    }
}
```

```

public function getName()
{
    return $this->name;
}

public function setPrice($price)
{
    if ($price < 0) {
        throw new \InvalidArgumentException('Цена не может быть отрицательной');
    }
    $this->price = $price;
}

public function getPrice()
{
    //Преобразование значение в double (формат числа с плавающей запятой), прежде чем использовать
    return (double) $this->price;
}

```

Публичные свойства облегчают создание кода. Напротив, применение getters/setters делает ваш код тестируемым, расширяемым и удобным в сопровождении. Разработчик вправе сам определить способ описания модели. ORM совместим с обоими способами.

Прим. переводчика : В то же время, использование getters/setters позволяет использовать некоторые преимущества такого способа. Например, если модель имеет связь один-ко-многим с другой моделью, при запросе связанной модели будет произведено N+1 запросов к базе данных. Напротив, при использовании getters/setters модель сделает только 2 запроса.

```

<?php

class Robots extends \Phalcon\Mvc\Model
{

    protected $id;

    protected $name;

    public function getId()
    {
        return $this->id;
    }

    public function setName($name)
    {
        if (strlen($name) < 10) {
            throw new \InvalidArgumentException('Имя слишком короткое');
        }
        $this->name = $name;
    }

    public function getName()
    {
        return $this->name;
    }

    public function initialize()
    {

```

```
        $this->hasMany("id", "RobotsParts", "robots_id");
    }

    /**
     * Возвращаем "robots parts" одним запросом
     *
     * @return \RobotsParts[]
     */
    public function getRobotsParts($parameters=null)
    {
        return $this->getRelated('RobotsParts', $parameters);
    }

}
```

## 2.11.2 Модели в Пространствах Имен

Пространства имен могут быть использованы во избежание конфликтов, связанных с именами классов. В этом случае, имя таблицы, из которой модель получает данные, соответствует имени класса (преобразуется в нижний регистр).

```
<?php

namespace Store\Toys;

class Robots extends \Phalcon\Mvc\Model
{



}
```

## 2.11.3 Понимание Записей В Объектах

Каждый экземпляр объекта модели представляет собой строку таблицы базы данных. Вы можете легко получить доступ к любой записи, считывая свойство объекта. К примеру, для таблицы “robots” с записями:

```
mysql> select * from robots;
+---+-----+-----+-----+
| id | name      | type      | year   |
+---+-----+-----+-----+
| 1 | Robotina  | mechanical | 1972 |
| 2 | Astro Boy | mechanical | 1952 |
| 3 | Terminator | cyborg    | 2029 |
+---+-----+-----+-----+
3 строки в наборе (0,00 сек)
```

Вы можете найти определенную запись по ее первичному ключу и напечатать ее имя:

```
<?php

// Найти запись с id = 3
$robot = Robots::findFirst(3);

// Печатать "Terminator"
echo $robot->name;
```

Как только запись будет зарезервирована в памяти, мы можем производить изменения ее данных, а затем сохранить изменения.

```
<?php

$robot = Robots::findFirst(3);
$robot->name = "RoboCop";
$robot->save();
```

Как вы можете видеть, нет никакой необходимости в использовании необработанных SQL запросов. *Phalcon|Mvc|Model* предоставляет высший уровень абстракции базы данных для веб-приложений.

## 2.11.4 Поиск записей

*Phalcon|Mvc|Model* также предлагает несколько методов для выборки записей. В следующем примере мы покажем вам как запросить одну или несколько записей из модели:

```
<?php

// Сколько роботов есть?
$robots = Robots::find();
echo "There are ", count($robots), "\n";

// Сколько существует механических роботов?
$robots = Robots::find("type = 'mechanical'");
echo "There are ", count($robots), "\n";

// Получить и распечатать виртуальных роботов упорядоченные по имени
$robots = Robots::find(array(
    "type = 'virtual'",
    "order" => "name"
));
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

// Получить первые 100 виртуальных роботов упорядоченных по имени
$robots = Robots::find(array(
    "type = 'virtual'",
    "order" => "name",
    "limit" => 100
));
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}
```

Вы также можете использовать метод `findFirst()`, чтобы получить только первую запись для данного критерия:

```
<?php

// Первый робот в таблице роботов
$robot = Robots::findFirst();
echo "The robot name is ", $robot->name, "\n";

// Первый механический робот в таблице роботов
$robot = Robots::findFirst("type = 'mechanical'");
echo "The first mechanical robot name is ", $robot->name, "\n";
```

```
// Первый виртуальный робот упорядоченный по имени в таблице роботов
$robot = Robots::findFirst(array("type" = 'virtual', "order" => "name"));
echo "The first virtual robot name is ", $robot->name, "\n";
```

Оба метода find() и findFirst() принимают ассоциативный массив, определяющий критерии поиска:

```
<?php

$robot = Robots::findFirst(array(
    "type" = 'virtual',
    "order" => "name DESC",
    "limit" => 30
));

$robots = Robots::find(array(
    "conditions" => "type = ?1",
    "bind"        => array(1 => "virtual")
));
```

Доступные параметры запроса:

Существует еще один вариант записи запросов поиска, в объектно-ориентированном стиле:

```
<?php

$robots = Robots::query()
    ->where("type = :type:")
    ->andWhere("year < 2000")
    ->bind(array("type" => "mechanical"))
    ->order("name")
    ->execute();
```

Статический метод query() возвращает *Phalcon\ Mvc\ Model\ Criteria* объект, который нормально работает с автокомпилитом среды разработки.

Все запросы внутри обрабатываются как *PHQL* запросы. PHQL это высокоуровневый, объектно-ориентированный, SQL подобный язык. Этот язык предоставит вам больше возможностей для выполнения запросов, таких как объединение с другими моделями, определение группировок, добавление агрегации и т.д.

## Возвращение результатов моделью

В то время как findFirst() возвращает непосредственно экземпляр вызванного класса (когда это возвращаемые данные), метод find() возвращает *Phalcon\ Mvc\ Model\ Resultset\ Simple*. Этот объект включает в себя весь функционал такой как, обходы, поиск определенных записей, подсчет и прочее.

Эти объекты являются более мощными, чем стандартные массивы. Одна из важнейших особенностей *Phalcon\ Mvc\ Model\ Resultset* является то, что в любой момент времени, в памяти, есть только одна запись. Это очень помогает в управлении памятью особенно при работе с большими объемами данных.

```
<?php

// Получить всех роботов
$robots = Robots::find();

// Обход в foreach
foreach ($robots as $robot) {
    echo $robot->name, "\n";
```

```

}

// Обход в while
$robots->rewind();
while ($robots->valid()) {
    $robot = $robots->current();
    echo $robot->name, "\n";
    $robots->next();
}

// Посчитать количество роботов
echo count($robots);

// Альтернативный способ посчитать количество записей
echo $robots->count();

// Перемещение внутреннего курсора к третьему роботу
$robots->seek(2);
$robot = $robots->current()

// Access a robot by its position in the resultset
$robot = $robots[5];

// Доступ робота по его положению в наборе результатов
if (isset($robots[3])) {
    $robot = $robots[3];
}

// Получить первую запись в наборе результатов
$robot = $robots->getFirst();

// Получить последнюю запись
$robot = $robots->getLast();

```

Набор результатов в Phalcon эмулирует перемещение курсора, вы можете получить любую строку указав её позицию или найти внутренний указатель для определенной позиции. Обратите внимание, что некоторые системы баз данных не поддерживают курсоры с прокруткой, это заставляет базу данных повторно выполнить запрос для того, чтобы перемотать курсор в начало и получить запись в нужную позицию. Аналогично, если набор результатов вызывается несколько раз, запрос должен быть выполнен такое же количество раз.

Хранение больших результатов запроса в памяти может потребовать много ресурсов, из-за этого наборы результатов получаются из базы данных блоками по 32 строк снижая потребность в повторном выполнении запроса в ряде случаев экономя память.

Обратите внимание, что наборы результатов могут быть сериализованы и хранятся в кэше бэкэнда. *Phalcon\|Cache* может помочь с этой задачей. Тем не менее, сериализация данных вызывает *Phalcon\|Mvc\|Model* для получения всех данных из базы данных в массив, таким образом, потребление памяти увеличивается.

```

<?php

// Запрос всех записей из модели Parts
$parts = Parts::find();

// Сериализуем результат и сохраняем в файл
file_put_contents("cache.txt", serialize($parts));

```

```
// Достаём Parts из файла
$parts = unserialize(file_get_contents("cache.txt"));

// Обходим parts в foreach
foreach ($parts as $part) {
    echo $part->id;
}
```

## Привязка параметров

Привязка параметров также поддерживается в *Phalcon|Mvc|Model*. Использование привязки параметров рекомендуется, чтобы исключить возможность SQL инъекции. Привязка параметров поддерживает строки и числа.

```
<?php

// Запрос роботов с связывающими параметрами с строковыми заполнителями
$conditions = "name = :name: AND type = :type:";

//Параметры с ключом, названияя которого идентично заполнителю
$parameters = array(
    "name" => "Robotina",
    "type" => "maid"
);

//Выполнение запроса
$robots = Robots::find(array(
    $conditions,
    "bind" => $parameters
));

// Запрос роботов с связывающими параметрами с числовыми заполнителями
$conditions = "name = ?1 AND type = ?2";
$parameters = array(1 => "Robotina", 2 => "maid");
$robots     = Robots::find(array(
    $conditions,
    "bind" => $parameters
));

// Запрос роботов с связывающими параметрами с строковыми и числовыми заполнителями
$conditions = "name = :name: AND type = ?1";

//Параметры с ключом, номер или название которого идентично заполнителем
$parameters = array(
    "name" => "Robotina",
    1 => "maid"
);

//Выполнение запроса
$robots = Robots::find(array(
    $conditions,
    "bind" => $parameters
));
```

При использовании цифровых указателей, необходимо определить их как целые числа, то есть 1 или 2. В этом случае “1” или “2” считаются строками, поэтому указатель не может быть успешно заменен. Строки автоматически изолируются используя PDO. Эта функция принимает во внимание кодиров-

ку соединения с базой данных, поэтому её рекомендуется определять в параметрах соединения или в конфигурации базы данных, неправильная кодировка будет приводить к некорректному хранению и извлечению данных. Кроме того, вы можете установить параметр “bindTypes”, что позволит определить, каким образом параметры должны быть связаны в соответствии с его типом данных:

```
<?php

use \Phalcon\Db\Column;

//Привязка параметров
$parameters = array(
    "name" => "Robotina",
    "year" => 2008
);

//Привязка типов параметров
$types = array(
    "name" => Column::BIND_PARAM_STR,
    "year" => Column::BIND_PARAM_INT
);

// Запрос роботов с связывающими параметрами и типами строковых заполнителей
$robots = Robots::find(array(
    "name = :name: AND year = :year:",
    "bind" => $parameters,
    "bindTypes" => $types
));

```

Поскольку тип-связывания по умолчанию \Phalcon\Db\Column::BIND\_PARAM\_STR, нет необходимости указывать параметр “bindTypes”, если все столбцы этого типа.

Привязка параметров доступна для всех запросов метода, таких как find() и findFirst(), а так же для методов count(), sum(), average() и т.д.

## 2.11.5 Инициализация/Изменение полученных записей

Может быть так, что вам необходимо произвести некоторые манипуляции с полученными записями. Для этого вы можете реализовать метод ‘afterFetch’ в модели. Этот метод выполняется каждый раз, когда экземпляр модели получает записи.

```
<?php

class Robots extends Phalcon\Mvc\Model
{

    public $id;

    public $name;

    public $status;

    public function beforeSave()
    {
        //Convert the array into a string
        $this->status = join(',', $this->status);
    }

    public function afterFetch()

```

```
{  
    //Convert the string to an array  
    $this->status = explode(',', $this->status);  
}  
}
```

Независимо от того, используете вы getters/setters или публичные свойства, вы можете реализовать обработку поля при получении доступа к последнему:

```
<?php  
  
class Robots extends Phalcon\Mvc\Model  
{  
    public $id;  
  
    public $name;  
  
    public $status;  
  
    public function getStatus()  
    {  
        return explode(',', $this->status);  
    }  
}
```

## 2.11.6 Отношения между моделями

Существует четыре типа отношений: один-к-одному, один-ко-многим, многие-к-одному и многие-ко-многим. Отношения могут быть односторонними или двунаправленными, и каждое может быть простым (один модель к одной) или более сложные (комбинация моделей). Модель менеджер управляет ограничением внешних ключей для этих отношений, их определение помогает ссылочной целостности, а также обеспечивает легкий и быстрый доступ к соответствующей записи в модели. Благодаря реализации отношений, легко получить доступ к данным в связанных моделях для любой выбранной записи(-ей).

### Односторонние отношения

Односторонние отношения это те отношения, которые генерируются в отношении друг к друга, но не наоборот.

### Двунаправленные отношения

Двунаправленные отношения создают отношения в обеих моделях, и каждая модель определяет обратную связь от другой.

### Определение отношений

В Phalcon, отношения должны быть определены в методе `initialize()` модели. Методы `belongsTo()`, `hasOne()` or `hasMany()` определяют отношения между одним или несколькими полями из текущей модели в поля другой модели. Каждый из этих методов требует 3 параметра: local fields, referenced model, referenced fields.

| Метод         | Описание                 |
|---------------|--------------------------|
| hasMany       | Определяет 1-n отношения |
| hasOne        | Определяет 1-1 отношения |
| belongsTo     | Определяет n-1 отношения |
| hasManyToMany | Определяет n-n отношения |

Следующая схема показывает 3 таблицы, чьи отношения будут служить нам в качестве примера, касающиеся отношений:

```
CREATE TABLE `robots` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `name` varchar(70) NOT NULL,
    `type` varchar(32) NOT NULL,
    `year` int(11) NOT NULL,
    PRIMARY KEY (`id`)
);

CREATE TABLE `robots_parts` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `robots_id` int(10) NOT NULL,
    `parts_id` int(10) NOT NULL,
    `created_at` DATE NOT NULL,
    PRIMARY KEY (`id`),
    KEY `robots_id` (`robots_id`),
    KEY `parts_id` (`parts_id`)
);

CREATE TABLE `parts` (
    `id` int(10) unsigned NOT NULL AUTO_INCREMENT,
    `name` varchar(70) NOT NULL,
    PRIMARY KEY (`id`)
);
```

- Модель “Robots” имеет несколько “RobotsParts”.
- Модель “Parts” имеет несколько “RobotsParts”.
- Модель “RobotsParts” принадлежит обоим “Robots” и “Parts” моделям как многие-к-одному.
- Модель “Robots” имеет отношение многие-ко-многим к “Parts” через “RobotsParts”

Посмотрим EER схему, чтобы лучше понять отношения:

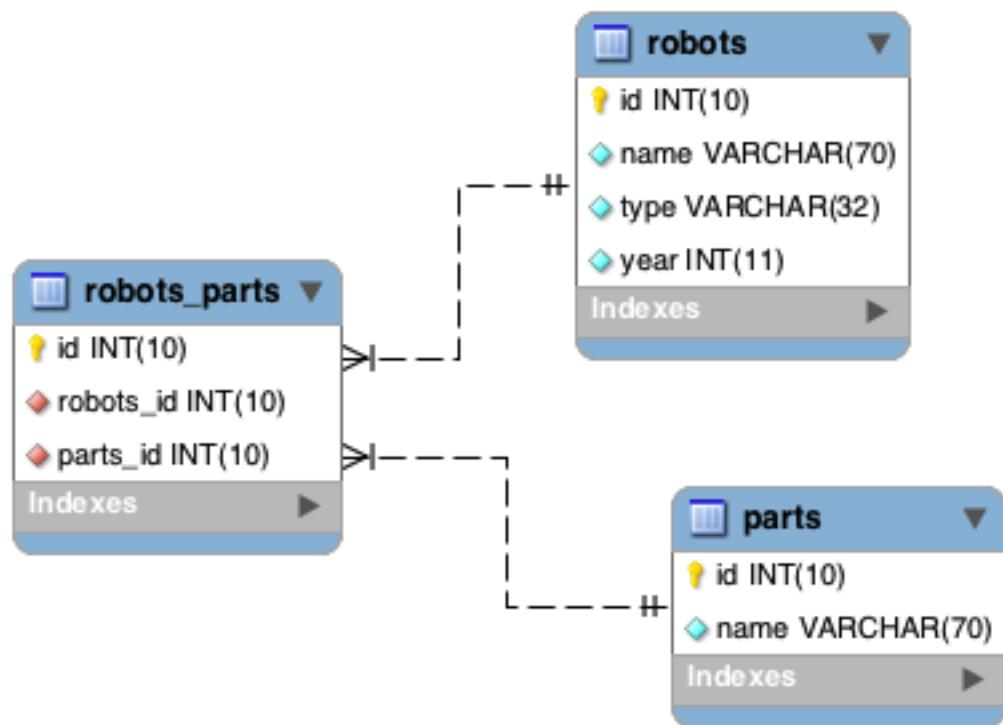
Модели с их отношениями могут быть реализованы следующим образом:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{
    public $id;

    public $name;

    public function initialize()
    {
        $this->hasMany("id", "RobotsParts", "robots_id");
    }
}
```



```
<?php

class Parts extends \Phalcon\Mvc\Model
{

    public $id;

    public $name;

    public function initialize()
    {
        $this->hasMany("id", "RobotsParts", "parts_id");
    }

}

<?php

class RobotsParts extends \Phalcon\Mvc\Model
{

    public $id;

    public $robots_id;

    public $parts_id;

    public function initialize()
    {
        $this->belongsTo("robots_id", "Robots", "id");
        $this->belongsTo("parts_id", "Parts", "id");
    }

}
```

Отношение “многие-ко-многим” требуют 3 модели и определение атрибутов, участвующих в отношениях:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{
    public $id;

    public $name;

    public function initialize()
    {
        $this->hasManyToMany(
            "id",
            "RobotsParts",
            "robots_id", "parts_id",
            "Parts",
            "id"
        );
    }
}
```

Первый параметр указывает локальные поля модели, используемые в отношениях; второй указывает имя модели и третью имя поля в указанной модели. Вы также можете использовать массивы для определения нескольких полей в отношениях.

## Преимущества отношений

При явном определении отношений между моделями, легко найти относящиеся записи для конкретной записи.

```
<?php
```

```
$robot = Robots::findFirst(2);
foreach ($robot->robotsParts as $robotPart) {
    echo $robotPart->parts->name, "\n";
}
```

Phalcon использует магические методы `__set`/`__get`/`__call` для сохранения или извлечения связанных данных, используя отношения.

По доступу к атрибуту с таким же именем, что и отношения, будем получать все связанные с ней записи.

```
<?php
```

```
$robot = Robots::findFirst();
$robotsParts = $robot->robotsParts; // все связанные записи с RobotsParts
```

Кроме того, вы можете использовать магию получателя:

```
<?php
```

```
$robot = Robots::findFirst();
$robotsParts = $robot->getRobotsParts(); // все связанные записи с RobotsParts
$robotsParts = $robot->getRobotsParts(array('limit' => 5)); // передача параметров
```

Если вызываемый метод “get” префикс `Phalcon\|Mvc\|Model` вернет `findFirst()`/`find()`. В следующем примере сравниваются получение соответствующих результатов с использованием магических методов и без:

```
<?php
```

```
$robot = Robots::findFirst(2);

// Модель Robots имеет отношение один-ко-многим 1-n (hasMany)
// Отношение к RobotsParts
$robotsParts = $robot->robotsParts;

// Только которые соответствуют условию
$robotsParts = $robot->getRobotsParts("created_at = '2012-03-15'");

// Или используя связанные параметры
$robotsParts = $robot->getRobotsParts(array(
    "created_at" = :date|,
    "bind" => array("date" => "2012-03-15")
));

$robotPart = RobotsParts::findFirst(1);
```

```
// Модель RobotsParts имеет отношение многие-к-одному n-1 (belongsTo)
// Отношение к Robots
$robot = $robotPart->robots;
```

Получение связанных записей вручную:

```
<?php

$robot = Robots::findFirst(2);

// Модель Robots имеет отношение один-ко-многим 1-n (hasMany)
// Отношение к RobotsParts
$robotsParts = RobotsParts::find("robots_id = '" . $robot->id . "'");

// Только которые соответствуют условиям
$robotsParts = $robotsParts::find(
    "robots_id = '" . $robot->id . "' AND created_at = '2012-03-15'"
);

$robotPart = RobotsParts::findFirst(1);

// Модель RobotsParts имеет отношение многие-к-одному n-1 (belongsTo)
// Отношение к RobotsParts
$robot = Robots::findFirst("id = '" . $robotPart->robots_id . "'");
```

Префикс “get” используется для find()/findFirst() связанных записей. В зависимости от типа отношений он будет использовать ‘find’ or ‘findFirst’:

| Тип              | Описание   | Неявный метод   |
|------------------|--|-----------------|
| Belongs-To       | Возвращает экземпляр модели взаимосвязанной записи   | findFirst       |
| Has-One          | Возвращает экземпляр модели взаимосвязанной записи   | findFirst       |
| Has-Many         | Возвращает коллекцию экземпляров модели для основной модели  | find            |
| Has-Many-to-Many | Returns a collection of model instances of the referenced model, it implicitly does ‘inner joins’ with the involved models | (complex query) |

Вы можете также использовать префикс “count” для подсчета количества связанных записей:

```
<?php

$robot = Robots::findFirst(2);
echo "The robot has ", $robot->countRobotsParts(), " parts\n";
```

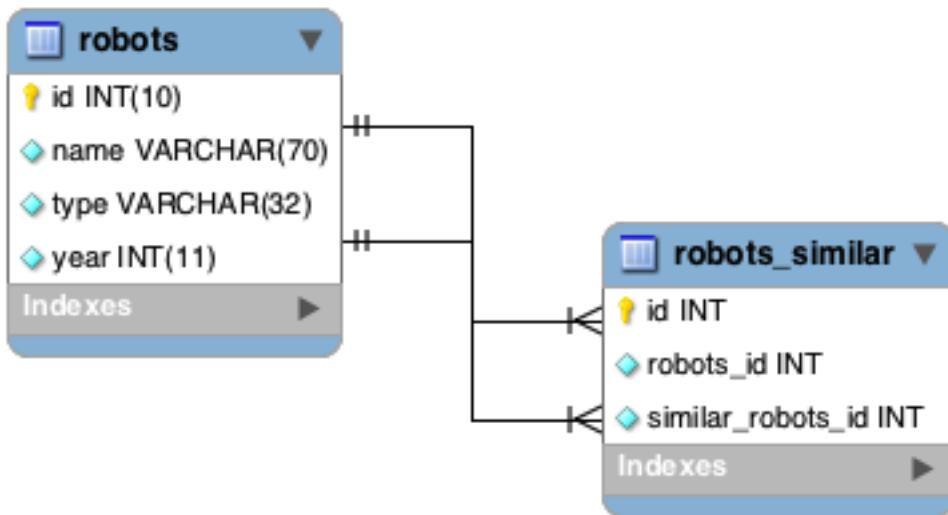
## Алиасы отношений

Чтобы лучше объяснить, как алиасы работают, давайте рассмотрим следующий пример:

В таблице “robots\_similar” есть функция, для определения, что роботы похожи на других:

```
mysql> desc robots_similar;
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id | int(10) unsigned | NO | PRI | NULL | auto_increment |
| robots_id | int(10) unsigned | NO | MUL | NULL | |
| similar_robots_id | int(10) unsigned | NO | | NULL | |
+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

Оба "robots\_id" и "similar\_robots\_id" имеют отношение к модели Robots:



Модель, которая отображает эту таблицу и ее отношения выглядит так:

```
<?php

class RobotsSimilar extends Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->belongsTo('robots_id', 'Robots', 'id');
        $this->belongsTo('similar_robots_id', 'Robots', 'id');
    }

}
```

Так как отношения указывают на ту же модель (Robots), получить записи, относящиеся к взаимосвязи корректно нельзя:

```
<?php

$robotsSimilar = RobotsSimilar::findFirst();

//Возвращает связанную запись на основе столбца (robots_id)
//Потому как имеет отношение belongsTo , это только возвращение одной записи
// но 'getRobots' , кажется, подразумевает, что вернётся больше, чем одна запись
$robot = $robotsSimilar->getRobots();

//но, как получить соответствующую запись на основании столбца (similar_robots_id)
//если оба отношения имеют одно и то же имя?
```

Алиасы позволяют переименовать оба отношения для решения этих проблем:

```
<?php

class RobotsSimilar extends Phalcon\Mvc\Model
{
```

```

public function initialize()
{
    $this->belongsTo('robots_id', 'Robots', 'id', array(
        'alias' => 'Robot'
    ));
    $this->belongsTo('similar_robots_id', 'Robots', 'id', array(
        'alias' => 'SimilarRobot'
    ));
}
}

```

С алиасами мы можем легко получить соответствующие записи:

```

<?php

$robotsSimilar = RobotsSimilar::findFirst();

//Возвращает связанную запись на основе столбца (robots_id)
$robot = $robotsSimilar->getRobot();
$robot = $robotsSimilar->robot;

//Возвращает связанную запись основанную на колонке (similar_robots_id)
$similarRobot = $robotsSimilar->getSimilarRobot();
$similarRobot = $robotsSimilar->similarRobot;

```

### Магические методы против явных

Большинство сред IDE и редакторов с авто-заполнением не могут определить правильность типов при использовании магических методов, вместо того, для получения удобства вы можете задать эти методы явно с соответствующим docblocks, помогая IDE для получения лучшего авто-завершения:

```

<?php

class Robots extends \Phalcon\Mvc\Model
{

    public $id;

    public $name;

    public function initialize()
    {
        $this->hasMany("id", "RobotsParts", "robots_id");
    }

    /**
     * Вернуться соотвествующий "robots parts"
     *
     * @return \RobotsParts[]
     */
    public function getRobotsParts($parameters=null)
    {
        return $this->getRelated('RobotsParts', $parameters);
    }
}

```

## 2.11.7 Виртуальные внешние ключи

По умолчанию, отношения не действуют как внешние ключи базы данных, то есть, если вы пытаетесь вставить/обновить значение, не имея действительного значения в эталонной модели, Phalcon не будет производить проверку сообщений. Вы можете изменить данное поведение, добавив четвертый параметр при определении отношения.

Модель RobotsPart может быть изменена, чтобы продемонстрировать эту функцию:

```
<?php

class RobotsParts extends \Phalcon\Mvc\Model
{

    public $id;

    public $robots_id;

    public $parts_id;

    public function initialize()
    {
        $this->belongsTo("robots_id", "Robots", "id", array(
            "foreignKey" => true
        ));

        $this->belongsTo("parts_id", "Parts", "id", array(
            "foreignKey" => array(
                "message" => "part_id не существует в модели Parts"
            )
        ));
    }
}
```

Если вы изменяете belongsTo() отношения в качестве внешнего ключа, он будет проверять, что значения вставляется/обновляется на тех полях где значение допустимое для эталонной модели. Аналогичным образом, если HasMany()/hasOne () изменяется он будет проверять, что записи не могут быть удалены, если эта запись используется для эталонной моделью.

```
<?php

class Parts extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->hasMany("id", "RobotsParts", "parts_id", array(
            "foreignKey" => array(
                "message" => "id не может быть удален, потому что используется в RobotsParts"
            )
        ));
    }
}
```

## Cascade/Ограничить действия

Отношения, которые действуют в качестве виртуальных внешних ключей по умолчанию ограничивают создание/обновление/удаление записей для поддержания целостности данных:

```
<?php

namespace Store\Models;

use Phalcon\Mvc\Model,
    Phalcon\Mvc\Model\Relation;

class Robots extends Model
{

    public $id;

    public $name;

    public function initialize()
    {
        $this->hasMany('id', 'Store\Models\Parts', 'robots_id', array(
            'foreignKey' => array(
                'action' => Relation::ACTION CASCADE
            )
        ));
    }
}
```

Приведенный выше код, удалит все относящиеся записи (parts), если основная запись (robot) удаляется.

### 2.11.8 Использование Расчетов

Расчеты являются помощниками для часто используемых функций СУБД, такие как COUNT, SUM, MAX, MIN или AVG. *Phalcon\ Mvc\ Model* позволяет использовать эти функции непосредственно с доступными методами.

Пример подсчета:

```
<?php

// Сколько сотрудников работает?
$rowcount = Employees::count();

// Сколько уникальных сфер деятельности рабочих?
$rowcount = Employees::count(array("distinct" => "area"));

// Сколько сотрудников работает в сфере тестирования?
$rowcount = Employees::count("area = 'Testing'");

// Количество сотрудников сгруппированных по сфере деятельности
$group = Employees::count(array("group" => "area"));
foreach ($group as $row) {
    echo $row->rowcount, " сотрудников в ", $row->area;
}

// Количество сотрудников сгруппированных по сфере деятельности упорядочено по их количеству
```

```
$group = Employees::count(array(
    "group" => "area",
    "order" => "rowcount"
));

// Избегайте SQL инъекции, используя связанные параметры
$group = Employees::count(array(
    "type > ?0"
    "bind" => array($type)
));

Пример суммы:

<?php

// Какая заработная плата всех сотрудников?
$total = Employees::sum(array("column" => "salary"));

// Какая заработная плата всех сотрудников в сфере продаж?
$total = Employees::sum(array(
    "column"      => "salary",
    "conditions" => "area = 'Sales'"
));

// Генерирует суммарную заработную плату каждой области
$group = Employees::sum(array(
    "column" => "salary",
    "group"  => "area"
));
foreach ($group as $row) {
    echo "Сумма заработной платы ", $row->area, " составляет ", $row->sumatory;
}

// Группирует зарплаты каждой сферы деятельности и упорядочивает их от большего к меньшему
$group = Employees::sum(array(
    "column" => "salary",
    "group"  => "area",
    "order"  => "sumatory DESC"
));

// Избегайте SQL инъекции, используя связанные параметры
$group = Employees::sum(array(
    "conditions" => "area > ?0"
    "bind" => array($area)
));
```

Пример поиска среднего:

```
<?php

// Какая средняя зарплата среди всех сотрудников?
$average = Employees::average(array("column" => "salary"));

// Какая средняя зарплата среди сотрудников сферы продаж?
$average = Employees::average(array(
    "column" => "salary",
    "conditions" => "area = 'Sales'"
));
```

```
// Избегайте SQL инъекции, используя связанные параметры
$average = Employees::average(array(
    "column" => "age"
    "conditions" => "area > ?0"
    "bind" => array($area)
));
```

Пример нахождения максимального/минимального:

```
<?php

// Какой максимальный возраст среди всех сотрудников?
$age = Employees::maximum(array("column" => "age"));

// Какой максимальный возраст среди сотрудников сферы продаж?
$age = Employees::maximum(array(
    "column" => "age",
    "conditions" => "area = 'Sales'"
));

// Какая минимальная зарплата среди сотрудников?
$salary = Employees::minimum(array("column" => "salary"));
```

## 2.11.9 Режимы гидратации

Как упоминалось выше, результирующие данные являются наборами комплексных объектов, это означает, что каждый возвращенный результат является объектом, представляющим собой строку в базе данных. Эти объекты могут быть изменены и сохранены снова :

```
<?php

// Изменение и сохранение полученных объектов модели роботов
foreach (Robots::find() as $robot) {
    $robot->year = 2000;
    $robot->save();
}
```

Иногда записи могут быть представлены пользователю в режиме только для чтения, это может быть полезно, чтобы изменить способ, в котором записи представлены для облегчения их обработки. Способ, используемый для представления объектов, возвращаемых в наборе результатов называется ‘режим гидратации’ :

```
<?php

use Phalcon\Mvc\Model\Resultset;

$robots = Robots::find();

// Вернёт каждого робота в виде массива
$robots->setHydrateMode(Resultset::HYDRATE_ARRAYS);

foreach ($robots as $robot) {
    echo $robot['year'], PHP_EOL;
}

// Вернёт каждого робота в stdClass
$robots->setHydrateMode(Resultset::HYDRATE_OBJECTS);
```

```
foreach ($robots as $robot) {
    echo $robot->year, PHP_EOL;
}

//Вернёт каждого робота как экземпляр объекта Robots
$robots->setHydrateMode(Resultset::HYDRATE_RECORDS);

foreach ($robots as $robot) {
    echo $robot->year, PHP_EOL;
}
```

Режим гидратации также может быть передан в качестве параметра в ‘find’:

```
<?php

use Phalcon\Mvc\Model\Resultset;

$robots = Robots::find(array(
    'hydration' => Resultset::HYDRATE_ARRAYS
));

foreach ($robots as $robot) {
    echo $robot['year'], PHP_EOL;
}
```

## 2.11.10 Создание/Обновление записей

Метод `Phalcon\ Mvc\ Model::save()` позволяет создавать/обновлять записи в зависимости от того, существуют ли они уже в таблице, связанной с моделью. Метод `save` вызывает методы `create` и `update` родительского класса `Phalcon\ Mvc\ Model`. Чтобы это работало, как и ожидалось, необходимо определить первичный ключ в таблице, чтобы определялось, запись должна быть создана или обновлена.

Также метод выполняет связанные валидаторы, виртуальные внешние ключи и события, которые определены в модели:

```
<?php

$robot      = new Robots();
$robot->type = "mechanical";
$robot->name = "Astro Boy";
$robot->year = 1952;
if ($robot->save() == false) {
    echo "Мы не можем сохранить робота прямо сейчас: \n";
    foreach ($robot->getMessages() as $message) {
        echo $message, "\n";
    }
} else {
    echo "Отлично, новый робот был успешно сохранен!";
}
```

В метод “`save`” может быть передан массив , чтобы избежать назначения каждому столбцу вручную. `Phalcon\ Mvc\ Model` будет проверять, есть ли сеттеры, реализованные для столбцов, для значений переданных в массиве, отдавая приоритет им вместо назначения значений непосредственно свойствам: ..

code-block:: php

```
<?php
$robot = new Robots(); $robot->save(array(
```

```
"type" => "mechanical", "name" => "Astro Boy", "year" => 1952
));
```

Значения назначенные непосредственно через атрибуты или через массив экранируются /проверяется в соответствии с типом данных атрибута. Таким образом, вы можете передать ненадежный массив, не беспокоясь о возможных SQL инъекциях :

```
<?php
```

```
$robot = new Robots();
$robot->save($_POST);
```

Без мер предосторожности к переданным данным от пользователей позволяет злоумышленнику установить значение любого столбца базы данных. Используйте эту функцию, если вы хотите, чтобы пользователь мог добавлять/обновлять каждый столбец в модели, даже если этих полей нет в отправленной форме.

Вы можете передать дополнительный параметр в метод 'save', чтобы установить список полей, которые должны быть приняты во внимание при выполнении переданных пользователем значений:

```
<?php
```

```
$robot = new Robots();
$robot->save($_POST, array('name', 'type'));
```

## Создание/Обновление с уверенностью

When an application has a lot of competition, we could be expecting create a record but it is actually updated. This could happen if we use Phalcon\ Mvc\ Model:: save() to persist the records in the database. If we want to be absolutely sure that a record is created or updated, we can change the save() call with create() or update():

```
<?php
```

```
$robot      = new Robots();
$robot->type = "mechanical";
$robot->name = "Astro Boy";
$robot->year = 1952;

//This record only must be created
if ($robot->create() == false) {
    echo "Umh, We can't store robots right now: \n";
    foreach ($robot->getMessages() as $message) {
        echo $message, "\n";
    }
} else {
    echo "Great, a new robot was created successfully!";
}
```

These methods "create" and "update" also accept an array of values as parameter.

## Auto-generated identity columns

Some models may have identity columns. These columns usually are the primary key of the mapped table. *Phalcon\ Mvc\ Model* can recognize the identity column omitting it in the generated SQL INSERT, so the

database system can generate an auto-generated value for it. Always after creating a record, the identity field will be registered with the value generated in the database system for it:

```
<?php  
  
$robot->save();  
  
echo "The generated id is: ", $robot->id;
```

*Phalcon|Mvc|Model* is able to recognize the identity column. Depending on the database system, those columns may be serial columns like in PostgreSQL or auto\_increment columns in the case of MySQL.

PostgreSQL uses sequences to generate auto-numeric values, by default, Phalcon tries to obtain the generated value from the sequence “table\_field\_seq”, for example: robots\_id\_seq, if that sequence has a different name, the method “getSequenceName” needs to be implemented:

```
<?php  
  
class Robots extends \Phalcon\Mvc\Model  
{  
  
    public function getSequenceName()  
    {  
        return "robots_sequence_name";  
    }  
  
}
```

## Storing related records

Magic properties can be used to store a records and its related properties:

```
<?php  
  
// Create a robot  
$artist = new Artists();  
$artist->name = 'Shinichi Osawa';  
$artist->country = 'Japan';  
  
// Create an album  
$album = new Albums();  
$album->name = 'The One';  
$album->artist = $artist; //Assign the artist  
$album->year = 2008;  
  
//Save both records  
$album->save();
```

Saving a record and its related records in a has-many relation:

```
<?php  
  
// Get an existing artist  
$artist = Artists::findFirst('name = "Shinichi Osawa"');  
  
// Create an album  
$album = new Albums();  
$album->name = 'The One';
```

```

$album->artist = $artist;

$songs = array();

// Create a first song
$songs[0] = new Songs();
$songs[0]->name = 'Star Guitar';
$songs[0]->duration = '5:54';

// Create a second song
$songs[1] = new Songs();
$songs[1]->name = 'Last Days';
$songs[1]->duration = '4:29';

// Assign the songs array
$album->songs = $songs;

// Save the album + its songs
$album->save();

```

Saving the album and the artist at the same time implicitly makes use of a transaction so if anything goes wrong with saving the related records, the parent will not be saved either. Messages are passed back to the user for information regarding any errors.

## Validation Messages

*Phalcon|Mvc|Model* has a messaging subsystem that provides a flexible way to output or store the validation messages generated during the insert/update processes.

Each message consists of an instance of the class *Phalcon|Mvc|Model\Message*. The set of messages generated can be retrieved with the method `getMessages()`. Each message provides extended information like the field name that generated the message or the message type:

```

<?php

if ($robot->save() == false) {
    foreach ($robot->getMessages() as $message) {
        echo "Message: ", $message->getMessage();
        echo "Field: ", $message->getField();
        echo "Type: ", $message->getType();
    }
}

```

*Phalcon|Mvc|Model* can generate the following types of validation messages:

| Type                   | Description  |
|------------------------|--|
| PresenceOf             | Generated when a field with a non-null attribute on the database is trying to insert/update a null value                           |
| ConstraintViolation    | Generated when a field part of a virtual foreign key is trying to insert/update a value that doesn't exist in the referenced model |
| InvalidValue           | Generated when a validator failed because of an invalid value  |
| InvalidCreateAttribute | Produced when a record is attempted to be created but it already exists  |
| InvalidUpdateAttribute | Produced when a record is attempted to be updated but it doesn't exist   |

The method `getMessages()` can be overridden in a model to replace/translate the default messages generated automatically by the ORM:

```
<?php

class Robots extends Phalcon\Mvc\Model
{
    public function getMessages()
    {
        $messages = array();
        foreach (parent::getMessages() as $message) {
            switch ($message->getType()) {
                case 'InvalidCreateAttempt':
                    $messages[] = 'The record cannot be created because it already exists';
                    break;
                case 'InvalidUpdateAttempt':
                    $messages[] = 'The record cannot be updated because it already exists';
                    break;
                case 'PresenceOf':
                    $messages[] = 'The field ' . $message->getField() . ' is mandatory';
                    break;
            }
        }
        return $messages;
    }
}
```

## Events and Events Manager

Models allow you to implement events that will be thrown when performing an insert/update/delete. They help define business rules for a certain model. The following are the events supported by *Phalcon\ Mvc\ Model* and their order of execution:

| Operation              | Name                   | Can stop operation?      | Explanation   |
|------------------------|------------------------|--------------------------|---|
| Inserting/<br>Updating | beforeValidation       | YES                      | Is executed before the fields are validated for not nulls/empty strings or foreign keys   |
| Inserting              | beforeValidationCreate | YES                      | Is executed before the fields are validated for not nulls/empty strings or foreign keys when an insertion operation is being made |
| Updating               | beforeValidationUpdate | YES                      | Is executed before the fields are validated for not nulls/empty strings or foreign keys when an updating operation is being made  |
| Inserting/<br>Updating | validationFail         | YES<br>(already stopped) | Is executed after an integrity validator fails  |
| Inserting              | afterValidationCreate  | YES                      | Is executed after the fields are validated for not nulls/empty strings or foreign keys when an insertion operation is being made  |
| Updating               | afterValidationUpdate  | YES                      | Is executed after the fields are validated for not nulls/empty strings or foreign keys when an updating operation is being made   |
| Inserting/<br>Updating | validation             | YES                      | Is executed after the fields are validated for not nulls/empty strings or foreign keys  |
| Inserting/<br>Updating | beforeSave             | YES                      | Runs before the required operation over the database system   |
| Updating               | beforeUpdate           | YES                      | Runs before the required operation over the database system only when an updating operation is being made                         |
| Inserting              | beforeCreate           | YES                      | Runs before the required operation over the database system only when an inserting operation is being made                        |
| Updating               | afterUpdate            | NO                       | Runs after the required operation over the database system only when an updating operation is being made                          |
| Inserting              | afterCreate            | NO                       | Runs after the required operation over the database system only when an inserting operation is being made                         |
| Inserting/<br>Updating | Save                   | NO                       | Runs after the required operation over the database system  |

### Implementing Events in the Model's class

The easier way to make a model react to events is implement a method with the same name of the event in the model's class:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function beforeValidationOnCreate()
    {
        echo "This is executed before creating a Robot!";
    }

}
```

Events can be useful to assign values before performing an operation, for example:

```
<?php

class Products extends \Phalcon\Mvc\Model
{
```

```
public function beforeCreate()
{
    //Set the creation date
    $this->created_at = date('Y-m-d H:i:s');
}

public function beforeUpdate()
{
    //Set the modification date
    $this->modified_in = date('Y-m-d H:i:s');
}

}
```

## Using a custom Events Manager

Additionally, this component is integrated with *Phalcon\Events\Manager*, this means we can create listeners that run when an event is triggered.

```
<?php

use Phalcon\Mvc\Model,
    Phalcon\Events\Manager as EventsManager;

class Robots extends Model
{

    public function initialize()
    {

        $eventsManager = new EventsManager();

        //Attach an anonymous function as a listener for "model" events
        $eventsManager->attach('model', function($event, $robot) {
            if ($event->getType() == 'beforeSave') {
                if ($robot->name == 'Scooby Doo') {
                    echo "Scooby Doo isn't a robot!";
                    return false;
                }
            }
            return true;
        });

        //Attach the events manager to the event
        $this->setEventsManager($eventsManager);
    }

}
```

In the example given above, `EventsManager` only acts as a bridge between an object and a listener (the anonymous function). Events will be fired to the listener when 'robots' are saved:

```
<?php

$robot = new Robots();
$robot->name = 'Scooby Doo';
```

```
$robot->year = 1969;
$robot->save();
```

If we want all objects created in our application use the same EventsManager, then we need to assign it to the Models Manager:

```
<?php

//Registering the modelsManager service
$di->setShared('modelsManager', function() {

    $eventsManager = new \Phalcon\Events\Manager();

    //Attach an anonymous function as a listener for "model" events
    $eventsManager->attach('model', function($event, $model){

        //Catch events produced by the Robots model
        if (get_class($model) == 'Robots') {

            if ($event->getType() == 'beforeSave') {
                if ($model->name == 'Scooby Doo') {
                    echo "Scooby Doo isn't a robot!";
                    return false;
                }
            }
        }

        return true;
    });

    //Setting a default EventsManager
    $modelsManager = new ModelsManager();
    $modelsManager->setEventsManager($eventsManager);
    return $modelsManager;
});
```

If a listener returns false that will stop the operation that is executing currently.

## Implementing a Business Rule

When an insert, update or delete is executed, the model verifies if there are any methods with the names of the events listed in the table above.

We recommend that validation methods are declared protected to prevent that business logic implementation from being exposed publicly.

The following example implements an event that validates the year cannot be smaller than 0 on update or insert:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function beforeSave()
    {
        if ($this->year < 0) {
            echo "Year cannot be smaller than zero!";
        }
    }
}
```

```
        return false;
    }
}

}
```

Some events return false as an indication to stop the current operation. If an event doesn't return anything, [Phalcon | Mvc | Model](#) will assume a true value.

## Validating Data Integrity

[Phalcon | Mvc | Model](#) provides several events to validate data and implement business rules. The special “validation” event allows us to call built-in validators over the record. Phalcon exposes a few built-in validators that can be used at this stage of validation.

The following example shows how to use it:

```
<?php

use Phalcon\Mvc\Model\Validator\InclusionIn,
    Phalcon\Mvc\Model\Validator\Uniqueness;

class Robots extends \Phalcon\Mvc\Model
{

    public function validation()
    {

        $this->validate(new InclusionIn(
            array(
                "field"  => "type",
                "domain" => array("Mechanical", "Virtual")
            )
        ));

        $this->validate(new Uniqueness(
            array(
                "field"      => "name",
                "message"   => "The robot name must be unique"
            )
        ));

        return $this->validationHasFailed() != true;
    }
}
```

The above example performs a validation using the built-in validator “InclusionIn”. It checks the value of the field “type” in a domain list. If the value is not included in the method then the validator will fail and return false. The following built-in validators are available:

| Name         | Explanation   | Example                 |
|--------------|---|-------------------------|
| Presence     | Validates that a field's value isn't null or empty string. This validator is automatically added based on the attributes marked as not null on the mapped table | <a href="#">Example</a> |
| Email        | Validates that field contains a valid email format  | <a href="#">Example</a> |
| Exclusion    | Validates that a value is not within a list of possible values  | <a href="#">Example</a> |
| Inclusion    | Validates that a value is within a list of possible values  | <a href="#">Example</a> |
| Numerical    | Validates that a field has a numeric format   | <a href="#">Example</a> |
| Regex        | Validates that the value of a field matches a regular expression  | <a href="#">Example</a> |
| Uniqueness   | Validates that a field or a combination of a set of fields are not present more than once in the existing records of the related table                          | <a href="#">Example</a> |
| StringLength | Validates the length of a string  | <a href="#">Example</a> |
| Url          | Validates that a value has a valid URL format   | <a href="#">Example</a> |

In addition to the built-in validators, you can create your own validators:

```
<?php

use Phalcon\Mvc\Model\Validator,
    Phalcon\Mvc\Model\ValidatorInterface;

class MaxMinValidator extends Validator implements ValidatorInterface
{

    public function validate($model)
    {
        $field = $this->getOption('field');

        $min = $this->getOption('min');
        $max = $this->getOption('max');

        $value = $model->$field;

        if ($min <= $value && $value <= $max) {
            $this->appendMessage(
                "The field doesn't have the right range of values",
                $field,
                "MaxMinValidator"
            );
            return false;
        }
        return true;
    }
}
```

Adding the validator to a model:

```
<?php

class Customers extends \Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new MaxMinValidator(
            array(
                "field" => "price",
                "min" => 10,
                "max" => 100
            )
        ));
    }
}
```

```
        )
    );
    if ($this->validationHasFailed() == true) {
        return false;
    }
}
```

```
}
```

The idea of creating validators is make them reusable between several models. A validator can also be as simple as:

```
<?php

use Phalcon\Mvc\Model,
    Phalcon\Mvc\Model\Message;

class Robots extends Model
{

    public function validation()
    {
        if ($this->type == "Old") {
            $message = new Message(
                "Sorry, old robots are not allowed anymore",
                "type",
                "MyType"
            );
            $this->appendMessage($message);
            return false;
        }
        return true;
    }
}
```

## Avoiding SQL injections

Every value assigned to a model attribute is escaped depending of its data type. A developer doesn't need to escape manually each value before storing it on the database. Phalcon uses internally the `bound` parameters capability provided by PDO to automatically escape every value to be stored in the database.

```
mysql> desc products;
+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra          |
+-----+-----+-----+-----+
| id         | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| product_types_id | int(10) unsigned | NO   | MUL | NULL    |                |
| name        | varchar(70)     | NO   |      | NULL    |                |
| price        | decimal(16,2)    | NO   |      | NULL    |                |
| active       | char(1)        | YES  |      | NULL    |                |
+-----+-----+-----+-----+
5 rows in set (0.00 sec)
```

If we use just PDO to store a record in a secure way, we need to write the following code:

```
<?php

$productTypeId = 1;
$name = 'Artichoke';
$price = 10.5;
$active = 'Y';

$sql = 'INSERT INTO products VALUES (null, :productTypeId, :name, :price, :active)';
$stmt = $db->prepare($sql);

$stmt->bindParam(':productTypeId', $productTypeId, PDO::PARAM_INT);
$stmt->bindParam(':name', $name, PDO::PARAM_STR, 70);
$stmt->bindParam(':price', doubleval($price));
$stmt->bindParam(':active', $active, PDO::PARAM_STR, 1);

$stmt->execute();
```

The good news is that Phalcon do this for you automatically:

```
<?php

$product = new Products();
$product->product_types_id = 1;
$product->name = 'Artichoke';
$product->price = 10.5;
$product->active = 'Y';
$product->create();
```

### 2.11.11 Skipping Columns

To tell Phalcon\ Mvc\ Model that always omits some fields in the creation and/or update of records in order to delegate the database system the assignation of the values by a trigger or a default:

```
<?php

class Robots extends \Phalcon\ Mvc\ Model
{

    public function initialize()
    {
        //Skips fields/columns on both INSERT/UPDATE operations
        $this->skipAttributes(array('year', 'price'));

        //Skips only when inserting
        $this->skipAttributesOnCreate(array('created_at'));

        //Skips only when updating
        $this->skipAttributesOnUpdate(array('modified_in'));
    }
}
```

This will ignore globally these fields on each INSERT/UPDATE operation on the whole application. Forcing a default value can be done in the following way:

```
<?php
```

```
$robot = new Robots();
$robot->name = 'Bender';
$robot->year = 1999;
$robot->created_at = new \Phalcon\Db\RawValue('default');
$robot->create();
```

A callback also can be used to create a conditional assignment of automatic default values:

```
<?php

use Phalcon\Mvc\Model,
    Phalcon\Db\RawValue;

class Robots extends Model
{
    public function beforeCreate()
    {
        if ($this->price > 10000) {
            $this->type = new RawValue('default');
        }
    }
}
```

Never use a \Phalcon\Db\RawValue to assign external data (such as user input) or variable data. The value of these fields is ignored when binding parameters to the query. So it could be used to attack the application injecting SQL.

## Dynamic Update

SQL UPDATE statements are by default created with every column defined in the model (full all-field SQL update). You can change specific models to make dynamic updates, in this case, just the fields that had changed are used to create the final SQL statement.

In some cases this could improve the performance by reducing the traffic between the application and the database server, this specially helps when the table has blob/text fields:

```
<?php

class Robots extends Phalcon\Mvc\Model
{
    public function initialize()
    {
        $this->useDynamicUpdate(true);
    }
}
```

### 2.11.12 Deleting Records

The method Phalcon\Mvc\Model::delete() allows to delete a record. You can use it as follows:

```
<?php

$robot = Robots::findFirst(11);
if ($robot != false) {
    if ($robot->delete() == false) {
        echo "Sorry, we can't delete the robot right now: \n";
```

```

        foreach ($robot->getMessages() as $message) {
            echo $message, "\n";
        }
    } else {
        echo "The robot was deleted successfully!";
    }
}

```

You can also delete many records by traversing a resultset with a foreach:

```
<?php

foreach (Robots::find("type='mechanical'") as $robot) {
    if ($robot->delete() == false) {
        echo "Sorry, we can't delete the robot right now: \n";
        foreach ($robot->getMessages() as $message) {
            echo $message, "\n";
        }
    } else {
        echo "The robot was deleted successfully!";
    }
}

```

The following events are available to define custom business rules that can be executed when a delete operation is performed:

| Operation | Name         | Can stop operation? | Explanation                              |
|-----------|--------------|---------------------|--|
| Deleting  | beforeDelete | YES                 | Runs before the delete operation is made |
| Deleting  | afterDelete  | NO                  | Runs after the delete operation was made |

With the above events can also define business rules in the models:

```
<?php

class Robots extends Phalcon\Mvc\Model
{

    public function beforeDelete()
    {
        if ($this->status == 'A') {
            echo "The robot is active, it can't be deleted";
            return false;
        }
        return true;
    }

}

```

### 2.11.13 Validation Failed Events

Another type of events are available when the data validation process finds any inconsistency:

| Operation                | Name              | Explanation  |
|--------------------------|-------------------|--|
| Insert or Update         | notSave           | Triggered when the INSERT or UPDATE operation fails for any reason |
| Insert, Delete or Update | onValidationFails | Triggered when any data manipulation operation fails               |

## 2.11.14 Behaviors

Behaviors are shared conducts that several models may adopt in order to re-use code, the ORM provides an API to implement behaviors in your models. Also, you can use the events and callbacks as seen before as an alternative to implement Behaviors with more freedom.

A behavior must be added in the model initializer, a model can have zero or more behaviors:

```
<?php

use Phalcon\Mvc\Model\Behavior\Timestampable;

class Users extends \Phalcon\Mvc\Model
{
    public $id;

    public $name;

    public $created_at;

    public function initialize()
    {
        $this->addBehavior(new Timestampable(
            array(
                'beforeCreate' => array(
                    'field' => 'created_at',
                    'format' => 'Y-m-d'
                )
            )
        ));
    }
}
```

The following built-in behaviors are provided by the framework:

| Name          | Description  |
|---------------|--|
| Timestampable | Allows to automatically update a model's attribute saving the datetime when a record is created or updated |
| SoftDelete    | Instead of permanently delete a record it marks the record as deleted changing the value of a flag column  |

### Timestampable

This behavior receives an array of options, the first level key must be an event name indicating when the column must be assigned:

```
<?php

public function initialize()
{
    $this->addBehavior(new Timestampable(
        array(
            'beforeCreate' => array(
                'field' => 'created_at',
                'format' => 'Y-m-d'
            )
        )
    ));
}
```

```
    ));
}
```

Each event can have its own options, ‘field’ is the name of the column that must be updated, if ‘format’ is a string it will be used as format of the PHP’s function `date`, format can also be an anonymous function providing you the free to generate any kind timestamp:

```
<?php

public function initialize()
{
    $this->addBehavior(new Timestampable(
        array(
            'beforeCreate' => array(
                'field' => 'created_at',
                'format' => function() {
                    $datetime = new Datetime(new DateTimeZone('Europe/Stockholm'));
                    return $datetime->format('Y-m-d H:i:sP');
                }
            )
        )
    )));
}
```

If the option ‘format’ is omitted a timestamp using the PHP’s function `time`, will be used.

## SoftDelete

This behavior can be used in the following way:

```
<?php

use Phalcon\Mvc\Model\Behavior\SoftDelete;

class Users extends \Phalcon\Mvc\Model
{

    const DELETED = 'D';

    const NOT_DELETED = 'N';

    public $id;

    public $name;

    public $status;

    public function initialize()
    {
        $this->addBehavior(new SoftDelete(
            array(
                'field' => 'status',
                'value' => Users::DELETED
            )
        ));
    }
}
```

This behavior accepts two options: ‘field’ and ‘value’, ‘field’ determines what field must be updated and ‘value’ the value to be deleted. Let’s pretend the table ‘users’ has the following data:

```
mysql> select * from users;
+---+-----+-----+
| id | name   | status |
+---+-----+-----+
| 1  | Lana   | N     |
| 2  | Brandon | N     |
+---+-----+-----+
2 rows in set (0.00 sec)
```

If we delete any of the two records the status will be updated instead of delete the record:

```
<?php
```

```
Users::findFirst(2)->delete();
```

The operation will result in the following data in the table:

```
mysql> select * from users;
+---+-----+-----+
| id | name   | status |
+---+-----+-----+
| 1  | Lana   | N     |
| 2  | Brandon | D     |
+---+-----+-----+
2 rows in set (0.01 sec)
```

Note that you need to specify the deleted condition in your queries to effectively ignore them as deleted records, this behavior doesn’t support that.

## Creating your own behaviors

The ORM provides an API to create your own behaviors. A behavior must be a class implementing the [Phalcon\ Mvc\ Model\ BehaviorInterface](#). Also, Phalcon\ Mvc\ Model\ Behavior provides most of the methods needed to ease the implementation of behaviors.

The following behavior is an example, it implements the Blameable behavior which helps identify the user that is performed operations over a model:

```
<?php
```

```
use Phalcon\ Mvc\ Model\ Behavior,
    Phalcon\ Mvc\ Model\ BehaviorInterface;

class Blameable extends Behavior implements BehaviorInterface
{

    public function notify($eventType, $model)
    {
        switch ($eventType) {

            case 'afterCreate':
            case 'afterDelete':
            case 'afterUpdate':

                $userName = // ... get the current user from session
        }
    }
}
```

```

    //Store in a log the username - event type and primary key
    file_put_contents(
        'logs/blamable-log.txt',
        $userName . ' ' . $eventType . ' ' . $model->id
    );

    break;

    default:
        /* ignore the rest of events */
    }
}
}

```

The former is a very simple behavior, but it illustrates how to create a behavior, now let's add this behavior to a model:

```

<?php

class Profiles extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->addBehavior(new Blamable());
    }

}

```

A behavior is also capable of intercept missing methods on your models:

```

<?php

use Phalcon\Mvc\Model\Behavior,
    Phalcon\Mvc\Model\BehaviorInterface;

class Sluggable extends Behavior implements BehaviorInterface
{

    public function missingMethod($model, $method, $arguments=array())
    {
        // if the method is 'getSlug' convert the title
        if ($method == 'getSlug') {
            return Phalcon\Tag::friendlyTitle($model->title);
        }
    }

}

```

Call that method on a model that implements Sluggable returns a SEO friendly title:

```

<?php

$title = $post->getSlug();

```

## Using Traits as behaviors

Starting from PHP 5.4 you can use [Traits](#) to re-use code in your classes, this is another way to implement custom behaviors. The following trait implements a simple version of the Timestampable behavior:

```
<?php

trait MyTimestampable
{

    public function beforeCreate()
    {
        $this->created_at = date('r');
    }

    public function beforeUpdate()
    {
        $this->updated_at = date('r');
    }

}
```

Then you can use it in your model as follows:

```
<?php

class Products extends \Phalcon\Mvc\Model
{
    use MyTimestampable;
}
```

## 2.11.15 Transactions

When a process performs multiple database operations, it is often that each step is completed successfully so that data integrity can be maintained. Transactions offer the ability to ensure that all database operations have been executed successfully before the data are committed to the database.

Transactions in Phalcon allow you to commit all operations if they have been executed successfully or rollback all operations if something went wrong.

### Manual Transactions

If an application only uses one connection and the transactions aren't very complex, a transaction can be created by just moving the current connection to transaction mode, doing a rollback or commit if the operation is successfully or not:

```
<?php

class RobotsController extends Phalcon\Mvc\Controller
{
    public function saveAction()
    {
        $this->db->begin();

        $robot = new Robots();
```

```

$robot->name = "WALL-E";
$robot->created_at = date("Y-m-d");
if ($robot->save() == false) {
    $this->db->rollback();
    return;
}

$robotPart = new RobotParts();
$robotPart->robots_id = $robot->id;
$robotPart->type = "head";
if ($robotPart->save() == false) {
    $this->db->rollback();
    return;
}

$this->db->commit();
}
}

```

## Implicit Transactions

Existing relationships can be used to store records and their related instances, this kind of operation implicitly creates a transaction to ensure that data are correctly stored:

```

<?php

$robotPart = new RobotParts();
$robotPart->type = "head";

$robot = new Robots();
$robot->name = "WALL-E";
$robot->created_at = date("Y-m-d");
$robot->robotPart = $robotPart;

$robot->save(); //Creates an implicit transaction to store both records

```

## Isolated Transactions

Isolated transactions are executed in a new connection ensuring that all the generated SQL, virtual foreign key checking and business rules are isolated from the main connection. This kind of transaction requires a transaction manager that globally manages each transaction created ensuring that it's correctly rollbacked/committed before ending the request:

```

<?php

use Phalcon\Mvc\Model\Transaction\Manager as TxManager,
    Phalcon\Mvc\Model\Transaction\Failed as TxFailed;

try {

    //Create a transaction manager
    $manager = new TxManager();

    // Request a transaction
    $transaction = $manager->get();

```

```
$robot = new Robots();
$robot->setTransaction($transaction);
$robot->name = "WALL·E";
$robot->created_at = date("Y-m-d");
if ($robot->save() == false) {
    $transaction->rollback("Cannot save robot");
}

$robotPart = new RobotParts();
$robotPart->setTransaction($transaction);
$robotPart->robots_id = $robot->id;
$robotPart->type = "head";
if ($robotPart->save() == false) {
    $transaction->rollback("Cannot save robot part");
}

//Everything goes fine, let's commit the transaction
$transaction->commit();

} catch(TxFailed $e) {
    echo "Failed, reason: ", $e->getMessage();
}
```

Transactions can be used to delete many records in a consistent way:

```
<?php

use Phalcon\Mvc\Model\Transaction\Manager as TxManager,
    Phalcon\Mvc\Model\Transaction\Failed as TxFailed;

try {

    //Create a transaction manager
    $manager = new TxManager();

    //Request a transaction
    $transaction = $manager->get();

    //Get the robots will be deleted
    foreach (Robots::find("type = 'mechanical'") as $robot) {
        $robot->setTransaction($transaction);
        if ($robot->delete() == false) {
            //Something goes wrong, we should to rollback the transaction
            foreach ($robot->getMessages() as $message) {
                $transaction->rollback($message->getMessage());
            }
        }
    }

    //Everything goes fine, let's commit the transaction
    $transaction->commit();

    echo "Robots were deleted successfully!";

} catch(TxFailed $e) {
    echo "Failed, reason: ", $e->getMessage();
}
```

Transactions are reused no matter where the transaction object is retrieved. A new transaction is generated only when a commit() or rollback() is performed. You can use the service container to create an overall transaction manager for the entire application:

```
<?php

$di->setShared('transactions', function(){
    return new \Phalcon\Mvc\Model\Transaction\Manager();
});
```

Then access it from a controller or view:

```
<?php

class ProductsController extends \Phalcon\Mvc\Controller
{

    public function saveAction()
    {

        //Obtain the TransactionsManager from the services container
        $manager = $this->di->getTransactions();

        //Or
        $manager = $this->transactions;

        //Request a transaction
        $transaction = $manager->get();

        //...
    }
}
```

While a transaction is active, the transaction manager will always return the same transaction across the application.

### 2.11.16 Independent Column Mapping

The ORM supports an independent column map, which allows the developer to use different column names in the model to the ones in the table. Phalcon will recognize the new column names and will rename them accordingly to match the respective columns in the database. This is a great feature when one needs to rename fields in the database without having to worry about all the queries in the code. A change in the column map in the model will take care of the rest. For example:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function columnMap()
    {
        //Keys are the real names in the table and
        //the values their names in the application
        return array(
            'id' => 'code',
            'the_name' => 'theName',
            'the_type' => 'theType',
        );
    }
}
```

```
        'the_year' => 'theYear'  
    );  
}  
  
}
```

Then you can use the new names naturally in your code:

```
<?php  
  
//Find a robot by its name  
$robot = Robots::findFirst("theName = 'Voltron'");  
echo $robot->theName, "\n";  
  
//Get robots ordered by type  
$robot = Robots::find(array('order' => 'theType DESC'));  
foreach ($robots as $robot) {  
    echo 'Code: ', $robot->code, "\n";  
}  
  
//Create a robot  
$robot = new Robots();  
$robot->code = '10101';  
$robot->theName = 'Bender';  
$robot->theType = 'Industrial';  
$robot->theYear = 2999;  
$robot->save();
```

Take into consideration the following the next when renaming your columns:

- References to attributes in relationships/validators must use the new names
- Refer the real column names will result in an exception by the ORM

The independent column map allow you to:

- Write applications using your own conventions
- Eliminate vendor prefixes/suffixes in your code
- Change column names without change your application code

## 2.11.17 Operations over Resultsets

If a resultset is composed of complete objects, the resultset is in the ability to perform operations on the records obtained in a simple manner:

### Updating related records

Instead of doing this:

```
<?php  
  
foreach ($robots->getParts() as $part) {  
    $part->stock = 100;  
    $part->updated_at = time();  
    if ($part->update() == false) {  
        foreach ($part->getMessages() as $message) {  
            echo $message;
```

```

        }
        break;
    }
}

```

you can do this:

```
<?php

$robots->getParts()->update(array(
    'stock' => 100,
    'updated_at' => time()
));
```

‘update’ also accepts an anonymous function to filter what records must be updated:

```
<?php

$data = array(
    'stock' => 100,
    'updated_at' => time()
);

//Update all the parts except those whose type is basic
$robots->getParts()->update($data, function($part) {
    if ($part->type == Part::TYPE_BASIC) {
        return false;
    }
    return true;
})
```

## Deleting related records

Instead of doing this:

```
<?php

foreach ($robots->getParts() as $part) {
    if ($part->delete() == false) {
        foreach ($part->getMessages() as $message) {
            echo $message;
        }
        break;
    }
}
```

you can do this:

```
<?php

$robots->getParts()->delete();
```

‘delete’ also accepts an anonymous function to filter what records must be deleted:

```
<?php

//Delete only whose stock is greater or equal than zero
$robots->getParts()->delete(function($part) {
```

```
if ($part->stock < 0) {
    return false;
}
return true;
});
```

### 2.11.18 Record Snapshots

Specific models could be set to maintain a record snapshot when they're queried. You can use this feature to implement auditing or just to know what fields are changed according to the data queried from the persistence:

```
<?php

class Robots extends Phalcon\Mvc\Model
{
    public function initialize()
    {
        $this->keepSnapshots(true);
    }
}
```

When activating this feature the application consumes a bit more of memory to keep track of the original values obtained from the persistence. In models that have this feature activated you can check what fields changed:

```
<?php

//Get a record from the database
$robot = Robots::findFirst();

//Change a column
$robot->name = 'Other name';

var_dump($robot->getChangedFields()); // ['name']
var_dump($robot->hasChanged('name')); // true
var_dump($robot->hasChanged('type'));
```

### 2.11.19 Models Meta-Data

To speed up development *Phalcon\ Mvc\ Model* helps you to query fields and constraints from tables related to models. To achieve this, *Phalcon\ Mvc\ Model\ MetaData* is available to manage and cache table meta-data.

Sometimes it is necessary to get those attributes when working with models. You can get a meta-data instance as follows:

```
<?php

$robot = new Robots();

// Get Phalcon\ Mvc\ Model\ Metadata instance
$metaData = $robot->getModelsMetaData();

// Get robots fields names
$attributes = $metaData->getAttributes($robot);
print_r($attributes);
```

```
// Get robots fields data types
$dataTypes = $metaData->getDataTypes($robot);
print_r($dataTypes);
```

## Caching Meta-Data

Once the application is in a production stage, it is not necessary to query the meta-data of the table from the database system each time you use the table. This could be done caching the meta-data using any of the following adapters:

| Adapter | Description   | API   |  |
|---------|---|---|--|
| Memory  | This adapter is the default. The meta-data is cached only during the request. When the request is completed, the meta-data are released as part of the normal memory of the request. This adapter is perfect when the application is in development so as to refresh the meta-data in each request containing the new and/or modified fields.     | <a href="#">Phalcon</a>   <a href="#">Mvc</a>   <a href="#">Model</a> | <a href="#">MetaData</a>   <a href="#">Memory</a>  |
| Session | This adapter stores meta-data in the <code>\$_SESSION</code> superglobal. This adapter is recommended only when the application is actually using a small number of models. The meta-data are refreshed every time a new session starts. This also requires the use of <code>session_start()</code> to start the session before using any models. | <a href="#">Phalcon</a>   <a href="#">Mvc</a>   <a href="#">Model</a> | <a href="#">MetaData</a>   <a href="#">Session</a> |
| Apc     | This adapter uses the <a href="#">Alternative PHP Cache (APC)</a> to store the table meta-data. You can specify the lifetime of the meta-data with options. This is the most recommended way to store meta-data when the application is in production stage.  | <a href="#">Phalcon</a>   <a href="#">Mvc</a>   <a href="#">Model</a> | <a href="#">MetaData</a>   <a href="#">Apc</a>     |
| XCache  | This adapter uses <a href="#">XCache</a> to store the table meta-data. You can specify the lifetime of the meta-data with options. This is the most recommended way to store meta-data when the application is in production stage.   | <a href="#">Phalcon</a>   <a href="#">Mvc</a>   <a href="#">Model</a> | <a href="#">MetaData</a>   <a href="#">Xcache</a>  |
| Files   | This adapter uses plain files to store meta-data. By using this adapter the disk-reading is increased but the database access is reduced  | <a href="#">Phalcon</a>   <a href="#">Mvc</a>   <a href="#">Model</a> | <a href="#">MetaData</a>   <a href="#">Files</a>   |

As other ORM's dependencies, the metadata manager is requested from the services container:

```
<?php

$di['modelsMetadata'] = function() {

    // Create a meta-data manager with APC
    $metaData = new \Phalcon\Mvc\Model\MetaData\Apc(array(
        "lifetime" => 86400,
        "prefix"   => "my-prefix"
    ));

    return $metaData;
};
```

## Meta-Data Strategies

As mentioned above the default strategy to obtain the model's meta-data is database introspection. In this strategy, the information schema is used to know the fields in a table, its primary key, nullable fields, data types, etc.

You can change the default meta-data introspection in the following way:

```
<?php

$di['modelsMetadata'] = function() {

    // Instantiate a meta-data adapter
    $metaData = new \Phalcon\Mvc\Model\MetaData\Apc(array(
        "lifetime" => 86400,
        "prefix"    => "my-prefix"
    ));

    //Set a custom meta-data introspection strategy
    $metaData->setStrategy(new MyIntrospectionStrategy());

    return $metaData;
};
```

## Database Introspection Strategy

This strategy doesn't require any customization and is implicitly used by all the meta-data adapters.

## Annotations Strategy

This strategy makes use of *annotations* to describe the columns in a model:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    /**
     * @Primary
     * @Identity
     * @Column(type="integer", nullable=false)
     */
    public $id;

    /**
     * @Column(type="string", length=70, nullable=false)
     */
    public $name;

    /**
     * @Column(type="string", length=32, nullable=false)
     */
    public $type;

    /**
     * @Column(type="integer", nullable=false)
     */
    public $year;

}
```

Annotations must be placed in properties that are mapped to columns in the mapped source. Properties without the @Column annotation are handled as simple class attributes.

The following annotations are supported:

| Name     | Description                                       |
|----------|---|
| Primary  | Mark the field as part of the table's primary key |
| Identity | The field is an auto_incremet/serial column       |
| Column   | This marks an attribute as a mapped column        |

The annotation @Column supports the following parameters:

| Name     | Description   |
|----------|---|
| type     | The column's type (string, integer, decimal, boolean) |
| length   | The column's length if any                            |
| nullable | Set whether the column accepts null values or not     |

The annotations strategy could be set up this way:

```
<?php

use Phalcon\Mvc\Model\MetaData\Apc as ApcMetaData,
    Phalcon\Meta\Strategy\Annotations as StrategyAnnotations;

$di['modelsMetadata'] = function() {

    // Instantiate a meta-data adapter
    $metaData = new ApcMetaData(array(
        "lifetime" => 86400,
        "prefix"   => "my-prefix"
    ));

    // Set a custom meta-data database introspection
    $metaData->setStrategy(new StrategyAnnotations());

    return $metaData;
};
```

## Manual Meta-Data

Phalcon can obtain the metadata for each model automatically without the developer must set them manually using any of the introspection strategies presented above.

The developer also has the option of define the metadata manually. This strategy overrides any strategy set in the meta-data manager. New columns added/modified/removed to/from the mapped table must be added/modified/removed also for everything to work properly.

The following example shows how to define the meta-data manually:

```
<?php

use Phalcon\Meta,
    Phalcon\Db\Column,
    Phalcon\Meta\MetaData;

class Robots extends Model
{

    public function metaData()
    {
        return array(
            // Every column in the mapped table
            MetaData::MODELS_ATTRIBUTES => array(
```

```
        'id', 'name', 'type', 'year'
    ),

    //Every column part of the primary key
MetaData::MODELS_PRIMARY_KEY => array(
    'id'
),

//Every column that isn't part of the primary key
MetaData::MODELS_NON_PRIMARY_KEY => array(
    'name', 'type', 'year'
),

//Every column that doesn't allow null values
MetaData::MODELS_NOT_NULL => array(
    'id', 'name', 'type', 'year'
),

//Every column and their data types
MetaData::MODELS_DATA_TYPES => array(
    'id' => Column::TYPE_INTEGER,
    'name' => Column::TYPE_VARCHAR,
    'type' => Column::TYPE_VARCHAR,
    'year' => Column::TYPE_INTEGER
),

//The columns that have numeric data types
MetaData::MODELS_DATA_TYPES_NUMERIC => array(
    'id' => true,
    'year' => true,
),

//The identity column, use boolean false if the model doesn't have
//an identity column
MetaData::MODELS_IDENTITY_COLUMN => 'id',

//How every column must be bound/casted
MetaData::MODELS_DATA_TYPES_BIND => array(
    'id' => Column::BIND_PARAM_INT,
    'name' => Column::BIND_PARAM_STR,
    'type' => Column::BIND_PARAM_STR,
    'year' => Column::BIND_PARAM_INT,
),

//Fields that must be ignored from INSERT SQL statements
MetaData::MODELS_AUTOMATIC_DEFAULT_INSERT => array(
    'year' => true
),

//Fields that must be ignored from UPDATE SQL statements
MetaData::MODELS_AUTOMATIC_DEFAULT_UPDATE => array(
    'year' => true
)

);
}
```

## 2.11.20 Pointing to a different schema

If a model is mapped to a table that is in a different schemas/databases than the default. You can use the `getSchema` method to define that:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function getSchema()
    {
        return "toys";
    }

}
```

## 2.11.21 Setting multiple databases

In Phalcon, all models can belong to the same database connection or have an individual one. Actually, when `Phalcon\ Mvc\ Model` needs to connect to the database it requests the “db” service in the application’s services container. You can overwrite this service setting it in the `initialize` method:

```
<?php

//This service returns a MySQL database
$di->set('dbMysql', function() {
    return new \Phalcon\Db\Adapter\Pdo\Mysql(array(
        "host" => "localhost",
        "username" => "root",
        "password" => "secret",
        "dbname" => "invo"
    ));
});

//This service returns a PostgreSQL database
$di->set('dbPostgres', function() {
    return new \Phalcon\Db\Adapter\Pdo\PostgreSQL(array(
        "host" => "localhost",
        "username" => "postgres",
        "password" => "",
        "dbname" => "invo"
    ));
});
```

Then, in the `Initialize` method, we define the connection service for the model:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->setConnectionService('dbPostgres');
    }

}
```

But Phalcon offers you more flexibility, you can define the connection that must be used to ‘read’ and for ‘write’. This is specially useful to balance the load to your databases implementing a master-slave architecture:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->setReadConnectionService('dbSlave');
        $this->setWriteConnectionService('dbMaster');
    }

}
```

The ORM also provides Horizontal Sharding facilities, by allowing you to implement a ‘shard’ selection according to the current query conditions:

```
<?php

class Robots extends Phalcon\Mvc\Model
{
    /**
     * Dynamically selects a shard
     *
     * @param array $intermediate
     * @param array $bindParams
     * @param array $bindTypes
     */
    public function selectReadConnection($intermediate, $bindParams, $bindTypes)
    {
        //Check if there is a 'where' clause in the select
        if (isset($intermediate['where'])) {

            $conditions = $intermediate['where'];

            //Choose the possible shard according to the conditions
            if ($conditions['left']['name'] == 'id') {
                $id = $conditions['right']['value'];
                if ($id > 0 && $id < 10000) {
                    return $this->getDI()->get('dbShard1');
                }
                if ($id > 10000) {
                    return $this->getDI()->get('dbShard2');
                }
            }
            //Use a default shard
            return $this->getDI()->get('dbShard0');
        }
    }
}
```

The method ‘selectReadConnection’ is called to choose the right connection, this method intercepts any new

query executed:

```
<?php

$robot = Robots::findFirst('id = 101');
```

## 2.11.22 Logging Low-Level SQL Statements

When using high-level abstraction components such as *Phalcon\ Mvc\ Model* to access a database, it is difficult to understand which statements are finally sent to the database system. *Phalcon\ Mvc\ Model* is supported internally by *Phalcon\ Db*. *Phalcon\ Logger* interacts with *Phalcon\ Db*, providing logging capabilities on the database abstraction layer, thus allowing us to log SQL statements as they happen.

```
<?php

use Phalcon\Logger,
    Phalcon\Db\Adapter\Pdo\Mysql as Connection,
    Phalcon\Events\Manager,
    Phalcon\Logger\Adapter\File;

$di->set('db', function() {

    $eventsManager = new EventsManager();

    $logger = new Logger("app/logs/debug.log");

    //Listen all the database events
    $eventsManager->attach('db', function($event, $connection) use ($logger) {
        if ($event->getType() == 'beforeQuery') {
            $logger->log($connection->getSQLStatement(), Logger::INFO);
        }
    });
});

$connection = new Connection(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "secret",
    "dbname" => "invo"
));

//Assign the eventsManager to the db adapter instance
$connection->setEventsManager($eventsManager);

return $connection;
});
```

As models access the default database connection, all SQL statements that are sent to the database system will be logged in the file:

```
<?php

$robot = new Robots();
$robot->name = "Robby the Robot";
$robot->created_at = "1956-07-21"
if ($robot->save() == false) {
    echo "Cannot save robot";
}
```

As above, the file `app/logs/db.log` will contain something like this:

```
[Mon, 30 Apr 12 13:47:18 -0500] [DEBUG] [Resource Id #77] INSERT INTO robots  
(name, created_at) VALUES ('Robby the Robot', '1956-07-21')
```

## 2.11.23 Profiling SQL Statements

Thanks to `Phalcon\Db`, the underlying component of `Phalcon\Mvc\Model`, it's possible to profile the SQL statements generated by the ORM in order to analyze the performance of database operations. With this you can diagnose performance problems and to discover bottlenecks.

```
<?php

$di->set('profiler', function(){
    return new \Phalcon\Db\Profiler();
}, true);

$di->set('db', function() use ($di) {

    $eventsManager = new \Phalcon\Events\Manager();

    //Get a shared instance of the DbProfiler
    $profiler = $di->getProfiler();

    //Listen all the database events
    $eventsManager->attach('db', function($event, $connection) use ($profiler) {
        if ($event->getType() == 'beforeQuery') {
            $profiler->startProfile($connection->getSQLStatement());
        }
        if ($event->getType() == 'afterQuery') {
            $profiler->stopProfile();
        }
    });
});

$connection = new \Phalcon\Db\Adapter\Pdo\Mysql(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "secret",
    "dbname" => "invo"
));

//Assign the eventsManager to the db adapter instance
$connection->setEventsManager($eventsManager);

return $connection;
});
```

Profiling some queries:

```
<?php

// Send some SQL statements to the database
Robots::find();
Robots::find(array("order" => "name"));
Robots::find(array("limit" => 30);

//Get the generated profiles from the profiler
$profiles = $di->get('profiler')->getProfiles();
```

```

foreach ($profiles as $profile) {
    echo "SQL Statement: ", $profile->getSQLStatement(), "\n";
    echo "Start Time: ", $profile->getInitialTime(), "\n";
    echo "Final Time: ", $profile->getFinalTime(), "\n";
    echo "Total Elapsed Time: ", $profile->getTotalElapsedSeconds(), "\n";
}

```

Each generated profile contains the duration in milliseconds that each instruction takes to complete as well as the generated SQL statement.

## 2.11.24 Injecting services into Models

You may be required to access the application services within a model, the following example explains how to do that:

```

<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function notSave()
    {
        //Obtain the flash service from the DI container
        $flash = $this->getDI()->getFlash();

        //Show validation messages
        foreach ($this->getMessages() as $message) {
            $flash->error($message);
        }
    }
}

```

The “notSave” event is triggered every time that a “create” or “update” action fails. So we’re flashing the validation messages obtaining the “flash” service from the DI container. By doing this, we don’t have to print messages after each save.

## 2.11.25 Disabling/Enabling Features

In the ORM we have implemented a mechanism that allow you to enable/disable specific features or options globally on the fly. According to how you use the ORM you can disable that you aren’t using. These options can also be temporarily disabled if required:

```

\Phalcon\Mvc\Model::setup(array(
    'events' => false,
    'columnRenaming' => false
));

```

The available options are:

| Option             | Description   | Default |
|--------------------|---|---------|
| events             | Enables/Disables callbacks, hooks and event notifications from all the models   | true    |
| columnRenaming     | Enables/Disables the column renaming  | true    |
| notNullValidations | The ORM automatically validate the not null columns present in the mapped table | true    |
| virtualForeignKeys | Enables/Disables the virtual foreign keys                                       | true    |
| phqlLiterals       | Enables/Disables literals in the PHQL parser                                    | true    |

## 2.11.26 Stand-Alone component

Using *Phalcon\ Mvc\ Model* in a stand-alone mode can be demonstrated below:

```
<?php

use Phalcon\DI,
    Phalcon\Db\Adapter\Pdo\Sqlite as Connection,
    Phalcon\Mvc\Model\Manager as ModelsManager,
    Phalcon\Mvc\Model\MetaData\Memory as MetaData,
    Phalcon\Mvc\Model;

$di = new DI();

//Setup a connection
$di->set('db', new Connection(array(
    "dbname" => "sample.db"
)));

//Set a models manager
$di->set('modelsManager', new ModelsManager());

//Use the memory meta-data adapter or other
$di->set('modelsMetadata', new MetaData());

//Create a model
class Robots extends Model
{

}

//Use the model
echo Robots::count();
```

## 2.12 Язык запросов Phalcon (PHQL)

Язык запросов phalcon, PhalconQL или просто PHQL — это высокоуровневый объектно-ориентированный диалект SQL, позволяющий писать запросы с использованием стандартизированного языка, похожего на SQL. PHQL реализован в виде парсера (написанного на C), который переводит синтаксис целевой СУБД.

Чтобы достигнуть максимально возможной производительности, Phalcon предоставляет парсер, используя технологию схожую с SQLite. Эта технология предоставляет небольшой парсер, который расходует малый объем памяти и, при этом, является поточно-ориентированным.

Сначала этот парсер проверяет синтаксис переданного выражения, затем строит его промежуточное представление и, в конце концов, преобразует его в SQL-диалект, соответствующий целевой СУБД.

В PHQL мы реализовали некоторый набор фич, чтобы ваш доступ к базе данных был более безопасным:

- Связанные (bound) параметры — часть языка PHQL, помогающая вам обезопасить ваш код
- PHQL разрешает выполнять только один SQL оператор за вызов, предотвращая инъекции
- PHQL игнорирует любые SQL-комментарии, часто использующиеся в SQL-инъекциях
- PHQL разрешает выполнять только операторы работы с данными, избегая изменения или удаления таблиц/баз данных по ошибке или извне без авторизации
- PHQL реализует высокую абстракцию, позволяющую вам оперировать моделями как таблицами и атрибутами класса — как полями таблицы.

### 2.12.1 Пример использования

Чтобы лучше объяснить, как работает PHQL рассмотрим следующий пример. У нас есть две модели, “Автомобили” и “Марки”:

```
<?php

class Cars extends Phalcon\Mvc\Model
{
    public $id;

    public $name;

    public $brand_id;

    public $price;

    public $year;

    public $style;

    /**
     * Эта модель отображается на таблицу sample_cars
     */
    public function getSource()
    {
        return 'sample_cars';
    }

    /**
     * Автомобиль может быть всего лишь одной марки, в то время как одну марку могут иметь множество автомобилей
     */
    public function initialize()
    {
        $this->belongsTo('brand_id', 'Brands', 'id');
    }
}
```

И каждый автомобиль имеет марку, в то время как у марки — множество автомобилей (в общем, “связь один ко многим”)

```
<?php

class Brands extends Phalcon\Mvc\Model
{

    public $id;

    public $name;

    /**
     * The model Brands is mapped to the "sample_brands" table
     */
    public function getSource()
    {
        return 'sample_brands';
    }

    /**
     * A Brand can have many Cars
     */
    public function initialize()
    {
        $this->hasMany('id', 'Cars', 'brand_id');
    }
}
```

## 2.12.2 Создание PHQL запросов

PHQL запросы могут быть созданы только как экземпляр класса *Phalcon\ Mvc\ Model\ Query*:

```
<?php

// Экземпляр Query
$query = new Phalcon\Mvc\Model\Query("SELECT * FROM Cars", $di);

// Выполнение запроса возвращает какой-то результат
$cars = $query->execute();
```

В контроллере или в представлении их проще создавать/выполнять используя внедрённый *models manager*:

```
<?php

// Исполнение простого запроса
$query = $this->modelsManager->createQuery("SELECT * FROM Cars");
$cars = $query->execute();

// Со связыванием (bound) параметров
$query = $this->modelsManager->createQuery("SELECT * FROM Cars WHERE name = :name:");
$cars = $query->execute(array(
    'name' => 'Audi'
));
```

Или еще проще:

```
<?php
```

```
// Исполнение простого запроса
$cars = $this->modelsManager->executeQuery("SELECT * FROM Cars");

// Со связыванием (bound) параметров
$cars = $this->modelsManager->executeQuery("SELECT * FROM Cars WHERE name = :name:", array(
    'name' => 'Audi',
));

```

### 2.12.3 Выборка записей

Как и в SQL, PHQL позволяет запрашивать записи используя оператор SELECT, с тем отличием, что вместо названий таблиц используются модели:

```
<?php

$query = $manager->createQuery("SELECT * FROM Cars ORDER BY Cars.name");
$query = $manager->createQuery("SELECT Cars.name FROM Cars ORDER BY Cars.name");
```

Так же разрешены неймспейсы классов:

```
<?php

$phql = "SELECT * FROM Formula\Cars ORDER BY Formula\Cars.name";
$query = $manager->createQuery($phql);

$phql = "SELECT Formula\Cars.name FROM Formula\Cars ORDER BY Formula\Cars.name";
$query = $manager->createQuery($phql);

$phql = "SELECT c.name FROM Formula\Cars c ORDER BY c.name";
$query = $manager->createQuery($phql);
```

PHQL поддерживает большинство стандартов SQL, даже такие нестандартные директивы как LIMIT:

```
<?php

$phql = "SELECT c.name FROM Cars AS c "
        . "WHERE c.brand_id = 21 ORDER BY c.name LIMIT 100";
$query = $manager->createQuery($phql);
```

#### Типы результата

Тип результата может меняться в зависимости от типа запрашиваемого нами столбца. При получении одного целого объекта, будет возвращён [Phalcon\ Mvc\ Model\ Resultset\ Simple](#). Этот вид результата представляет собой полноценный объект модели:

```
<?php

$phql = "SELECT c.* FROM Cars AS c ORDER BY c.name";
$cars = $manager->executeQuery($phql);
foreach ($cars as $car) {
    echo "Name: ", $car->name, "\n";
}
```

Это то же самое, что и:

```
<?php

$cars = Cars::find(array("order" => "name"));
foreach ($cars as $car) {
    echo "Name: ", $car->name, "\n";
}
```

Полноценные объекты могут быть изменены и пересохранены в базе данных, потому что они представляют собой полноценную запись в связанной таблице. Есть другие типы запросов, которые не возвращают такие объекты, например:

```
<?php

$phql = "SELECT c.id, c.name FROM Cars AS c ORDER BY c.name";
$cars = $manager->executeQuery($phql);
foreach ($cars as $car) {
    echo "Name: ", $car->name, "\n";
}
```

Тут мы запросили только некоторые поля таблицы, поэтому это не может являться объектом. Однако и в этом случае тоже возвращается *Phalcon\ Mvc\ Model\ Resultset\ Simple*. Но, тем не менее, каждый элемент выборки будет стандартным объектом, содержащим значения только двух запрошенных столбцов.

Такие значения, которые не представляют собой полноценного объекта, мы называем скалярами. PHQL позволяет вам запрашивать все типы скаляров: поля, функции, литералы, выражения и т.д.:

```
<?php

$phql = "SELECT CONCAT(c.id, ' ', c.name) AS id_name FROM Cars AS c ORDER BY c.name";
$cars = $manager->executeQuery($phql);
foreach ($cars as $car) {
    echo $car->id_name, "\n";
}
```

Раз уж мы можем запрашивать полноценные объекты и скаляры, то мы так же можем запросить их одновременно:

```
<?php

$phql = "SELECT c.price*0.16 AS taxes, c.* FROM Cars AS c ORDER BY c.name";
$result = $manager->executeQuery($phql);
```

В этом случае результатом будет объект *Phalcon\ Mvc\ Model\ Resultset\ Complex*. Он позволяет получить доступ и к полноценному объекту и к скаляру одновременно:

```
<?php

foreach ($result as $row) {
    echo "Name: ", $row->cars->name, "\n";
    echo "Price: ", $row->cars->price, "\n";
    echo "Taxes: ", $row->taxes, "\n";
}
```

Скаляры представлены как свойства каждой “row”, в то время как полноценные объекты — свойствами с названиями связанной модели.

## Джоины (Joins)

Используя PHQL очень просто запрашивать записи из нескольких моделей. Поддерживаются большинство различных джоинов. PHQL автоматически добавляет условия, которые мы определили при связывании моделей:

```
<?php
```

```
$phql = "SELECT Cars.name AS car_name, Brands.name AS brand_name FROM Cars JOIN Brands";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo $row->car_name, "\n";
    echo $row->brand_name, "\n";
}
```

По умолчанию используется INNER JOIN. Вы можете сами определить тип JOIN в запросе:

```
<?php
```

```
$phql = "SELECT Cars.*, Brands.* FROM Cars INNER JOIN Brands";
$rows = $manager->executeQuery($phql);

$phql = "SELECT Cars.*, Brands.* FROM Cars LEFT JOIN Brands";
$rows = $manager->executeQuery($phql);

$phql = "SELECT Cars.*, Brands.* FROM Cars LEFT OUTER JOIN Brands";
$rows = $manager->executeQuery($phql);

$phql = "SELECT Cars.*, Brands.* FROM Cars CROSS JOIN Brands";
$rows = $manager->executeQuery($phql);
```

Так же можно вручную задавать условия для JOIN'ов:

```
<?php
```

```
$phql = "SELECT Cars.*, Brands.* FROM Cars INNER JOIN Brands ON Brands.id = Cars.brands_id";
$rows = $manager->executeQuery($phql);
```

Джоины так же могут быть созданы, если в условии FROM фигурируют несколько таблиц:

```
<?php
```

```
$phql = "SELECT Cars.*, Brands.* FROM Cars, Brands WHERE Brands.id = Cars.brands_id";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo "Car: ", $row->cars->name, "\n";
    echo "Brand: ", $row->brands->name, "\n";
}
```

Если в запросе используется алиас для переименования модели, то это имя будет использовано для именования атрибутов в каждой строке результата:

```
<?php
```

```
$phql = "SELECT c.*, b.* FROM Cars c, Brands b WHERE b.id = c.brands_id";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo "Car: ", $row->c->name, "\n";
    echo "Brand: ", $row->b->name, "\n";
}
```

Когда присоединяемая модель имеет связь многие-ко-многим к ‘from’ модели, промежуточная модель неявно добавляется в сгенерированный запрос:

```
<?php

$phql = 'SELECT Brands.name, Songs.name FROM Artists .
    JOIN Songs WHERE Artists.genre = "Trip-Hop"';
$result = $this->modelsManager->query($phql);
```

Получаем следующий SQL в MySQL:

```
SELECT `brands`.`name`, `songs`.`name` FROM `artists`
INNER JOIN `albums` ON `albums`.`artists_id` = `artists`.`id`
INNER JOIN `songs` ON `albums`.`songs_id` = `songs`.`id`
WHERE `artists`.`genre` = 'Trip-Hop'
```

## Агрегаторы

Следующий пример показывает, как использовать агрегаторы в PHQL:

```
<?php

// Сколько стоят все машины?
$phql = "SELECT SUM(price) AS summatory FROM Cars";
$row = $manager->executeQuery($phql)->getFirst();
echo $row['summatory'];

// Сколько машин каждой марки?
$phql = "SELECT Cars.brand_id, COUNT(*) FROM Cars GROUP BY Cars.brand_id";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo $row->brand_id, ' ', $row["1"], "\n";
}

// Сколько различных марок?
$phql = "SELECT Brands.name, COUNT(*) FROM Cars JOIN Brands GROUP BY 1";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo $row->name, ' ', $row["1"], "\n";
}

$phql = "SELECT MAX(price) AS maximum, MIN(price) AS minimum FROM Cars";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo $row["maximum"], ' ', $row["minimum"], "\n";
}

// Сколько различных марок машин использовано?
$phql = "SELECT COUNT(DISTINCT brand_id) AS brandId FROM Cars";
$rows = $manager->executeQuery($phql);
foreach ($rows as $row) {
    echo $row->brandId, "\n";
}
```

## Условия

Условия позволяют отфильтровать необходимый нам набор записей для запроса. WHERE позволяет это сделать:

```
<?php

// Простые условия
$phql = "SELECT * FROM Cars WHERE Cars.name = 'Lamborghini Espada'";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.price > 10000";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE TRIM(Cars.name) = 'Audi R8'";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.name LIKE 'Ferrari%'";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.name NOT LIKE 'Ferrari%'";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.price IS NULL";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.id IN (120, 121, 122)";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.id NOT IN (430, 431)";
$cars = $manager->executeQuery($phql);

$phql = "SELECT * FROM Cars WHERE Cars.id BETWEEN 1 AND 100";
$cars = $manager->executeQuery($phql);
```

Так же, как часть PHQL, в целях безопасности, входные данные, переданные в качестве параметров, будут автоматически экранированы:

```
<?php

$phql = "SELECT * FROM Cars WHERE Cars.name = :name:";
$cars = $manager->executeQuery($phql, array("name" => 'Lamborghini Espada'));

$phql = "SELECT * FROM Cars WHERE Cars.name = ?0";
$cars = $manager->executeQuery($phql, array(0 => 'Lamborghini Espada'));
```

### 2.12.4 Вставка данных

С помощью PHQL можно вставлять данные используя знакомый уже оператор INSERT:

```
<?php

// Вставка без указания столбцов
$phql = "INSERT INTO Cars VALUES (NULL, 'Lamborghini Espada', "
    . "7, 10000.00, 1969, 'Grand Tourer')";
$manager->executeQuery($phql);
```

```
// Указание конкретных столбцов для вставки
$phql = "INSERT INTO Cars (name, brand_id, year, style) "
    . "VALUES ('Lamborghini Espada', 7, 1969, 'Grand Tourer')";
$manager->executeQuery($phql);

// Вставка с использованием плейсхолдеров
$phql = "INSERT INTO Cars (name, brand_id, year, style) "
    . "VALUES (:name:, :brand_id:, :year:, :style)";
$manager->executeQuery($sql,
    array(
        'name'      => 'Lamborghini Espada',
        'brand_id'  => 7,
        'year'       => 1969,
        'style'      => 'Grand Tourer',
    )
);
```

Phalcon не только преобразует PHQL выражения в SQL. Все события и бизнес-правила, определённые в модели будут выполнены, даже если мы создаём отдельные объекты вручную. Добавим правило в модель автомобилей, например, цена не может быть меньше \$ 10 000:

```
<?php

use Phalcon\Mvc\Model\Message;

class Cars extends Phalcon\Mvc\Model
{

    public function beforeCreate()
    {
        if ($this->price < 10000)
        {
            $this->appendMessage(new Message("A car cannot cost less than $ 10,000"));
            return false;
        }
    }
}
```

Теперь, если мы сделаем INSERT в модель Автомобилей, то эта операция не будет выполнена, потому что цену, которую мы передаем, не удовлетворяет реализованному правилу:

```
<?php

$phql    = "INSERT INTO Cars VALUES (NULL, 'Nissan Versa', 7, 9999.00, 2012, 'Sedan')";
$result = $manager->executeQuery($phql);
if ($result->success() == false)
{
    foreach ($result->getMessages() as $message)
    {
        echo $message->getMessage();
    }
}
```

## 2.12.5 Изменение данных

Изменение записей очень похоже на их вставку. Как вы знаете, для изменения данных используется UPDATE. Когда запись изменяется, события связанные с этой операцией вызываются для каждой записи.

```
<?php

// Изменение одного столбца
$phql = "UPDATE Cars SET price = 15000.00 WHERE id = 101";
$manager->executeQuery($phql);

// Изменение нескольких столбцов
$phql = "UPDATE Cars SET price = 15000.00, type = 'Sedan' WHERE id = 101";
$manager->executeQuery($phql);

// Изменение нескольких строк
$phql = "UPDATE Cars SET price = 7000.00, type = 'Sedan' WHERE brands_id > 5";
$manager->executeQuery($phql);

// Использование плейсхолдеров
$phql = "UPDATE Cars SET price = ?0, type = ?1 WHERE brands_id > ?2";
$manager->executeQuery($phql, array(
    0 => 7000.00,
    1 => 'Sedan',
    2 => 5
));
```

UPDATE выполняет изменение в два этапа:

- Сначала, если у UPDATE есть условия WHERE, извлекаются все записи подходящие под эти условия,
- Затем, на основе выбранных объектов их изменённые поля сохраняются в базе данных

Такой способ выполнения позволяет событиям, виртуальным внешним ключам и проверкам (validations) принять участие в процессе изменения данных. В итоге, вот такой код:

```
<?php

$phql = "UPDATE Cars SET price = 15000.00 WHERE id > 101";
$success = $manager->executeQuery($phql);
```

эквивалентен такому:

```
<?php

$messages = null;

$process = function() use (&$messages) {
    foreach (Cars::find("id > 101") as $car) {
        $car->price = 15000;
        if ($car->save() == false) {
            $messages = $car->getMessages();
            return false;
        }
    }
    return true;
};
```

```
$success = $process();
```

## 2.12.6 Удаление данных

Когда запись удаляется, события связанные с этой операцией будут выполнены для каждой записи:

```
<?php

// Удаление одной записи
$phql = "DELETE FROM Cars WHERE id = 101";
$manager->executeQuery($phql);

// Удаление нескольких записей
$phql = "DELETE FROM Cars WHERE id > 100";
$manager->executeQuery($phql);

// Использование плейсхолдеров
$phql = "DELETE FROM Cars WHERE id BETWEEN :initial: AND :final:";
$manager->executeQuery(
    $phql,
    array(
        'initial' => 1,
        'final' => 100
    )
);
```

Операция DELETE выполняется так же в два этапа, как и UPDATE.

## 2.12.7 Создание запросов с использованием Query Builder

Есть специальный конструктор для создания PHQL-запросов, избавляющий от необходимости писать PHQL-операторы и он так же весьма IDE-дружественен:

```
<?php

// Получение целого набора
$robots = $this->modelsManager->createBuilder()
    ->from('Robots')
    ->join('RobotsParts')
    ->order('Robots.name')
    ->getQuery()
    ->execute();

// Получение первой записи
$robots = $this->modelsManager->createBuilder()
    ->from('Robots')
    ->join('RobotsParts')
    ->order('Robots.name')
    ->getQuery()
    ->getSingleResult();
```

Что то же самое, что и:

```
<?php

$phql = "SELECT Robots.*
```

```

    FROM Robots JOIN RobotsParts p
    ORDER BY Robots.name LIMIT 20";
$result = $manager->executeQuery($phql);

```

Больше примеров использования конструктора:

```

<?php

$builder->from('Robots');
// 'SELECT Robots.* FROM Robots'

// 'SELECT Robots.*, RobotsParts.* FROM Robots, RobotsParts'
$builder->from(array('Robots', 'RobotsParts'));

// 'SELECT * FROM Robots'
$phql = $builder->columns('*')
    ->from('Robots');

// 'SELECT id FROM Robots'
$builder->columns('id')
    ->from('Robots');

// 'SELECT id, name FROM Robots'
$builder->columns(array('id', 'name'))
    ->from('Robots');

// 'SELECT Robots.* FROM Robots WHERE Robots.name = "Voltron"'
$builder->from('Robots')
    ->where('Robots.name = "Voltron"');

// 'SELECT Robots.* FROM Robots WHERE Robots.id = 100'
$builder->from('Robots')
    ->where(100);

// 'SELECT Robots.* FROM Robots WHERE Robots.type = "virtual" AND Robots.id > 50'
$builder->from('Robots')
    ->where('type = "virtual"')
    ->andWhere('id > 50');

// 'SELECT Robots.* FROM Robots WHERE Robots.type = "virtual" OR Robots.id > 50'
$builder->from('Robots')
    ->where('type = "virtual"')
    ->orWhere('id > 50');

// 'SELECT Robots.* FROM Robots GROUP BY Robots.name'
$builder->from('Robots')
    ->groupBy('Robots.name');

// 'SELECT Robots.* FROM Robots GROUP BY Robots.name, Robots.id'
$builder->from('Robots')
    ->groupBy(array('Robots.name', 'Robots.id'));

// 'SELECT Robots.name, SUM(Robots.price) FROM Robots GROUP BY Robots.name'
$builder->columns(array('Robots.name', 'SUM(Robots.price)'))
    ->from('Robots')
    ->groupBy('Robots.name');

// 'SELECT Robots.name, SUM(Robots.price) FROM Robots

```

```
// GROUP BY Robots.name HAVING SUM(Robots.price) > 1000'
$builder->columns(array('Robots.name', 'SUM(Robots.price)'))
    ->from('Robots')
    ->groupBy('Robots.name')
    ->having('SUM(Robots.price) > 1000');

// 'SELECT Robots.* FROM Robots JOIN RobotsParts';
$builder->from('Robots')
    ->join('RobotsParts');

// 'SELECT Robots.* FROM Robots JOIN RobotsParts AS p';
$builder->from('Robots')
    ->join('RobotsParts', null, 'p');

// 'SELECT Robots.* FROM Robots JOIN RobotsParts ON Robots.id = RobotsParts.robots_id AS p';
$builder->from('Robots')
    ->join('RobotsParts', 'Robots.id = RobotsParts.robots_id', 'p');

// 'SELECT Robots.* FROM Robots
// JOIN RobotsParts ON Robots.id = RobotsParts.robots_id AS p
// JOIN Parts ON Parts.id = RobotsParts.parts_id AS t'
$builder->from('Robots')
    ->join('RobotsParts', 'Robots.id = RobotsParts.robots_id', 'p')
    ->join('Parts', 'Parts.id = RobotsParts.parts_id', 't');

// 'SELECT r.* FROM Robots AS r'
$builder->addFrom('Robots', 'r');

// 'SELECT Robots.*, p.* FROM Robots, Parts AS p'
$builder->from('Robots')
    ->addFrom('Parts', 'p');

// 'SELECT r.*, p.* FROM Robots AS r, Parts AS p'
$builder->from(array('r' => 'Robots'))
    ->addFrom('Parts', 'p');

// 'SELECT r.*, p.* FROM Robots AS r, Parts AS p';
$builder->from(array('r' => 'Robots', 'p' => 'Parts'));

// 'SELECT Robots.* FROM Robots LIMIT 10'
$builder->from('Robots')
    ->limit(10);

// 'SELECT Robots.* FROM Robots LIMIT 10 OFFSET 5'
$builder->from('Robots')
    ->limit(10, 5);

// 'SELECT Robots.* FROM Robots WHERE id BETWEEN 1 AND 100'
$builder->from('Robots')
    ->betweenWhere('id', 1, 100);

// 'SELECT Robots.* FROM Robots WHERE id IN (1, 2, 3)'
$builder->from('Robots')
    ->inWhere('id', array(1, 2, 3));

// 'SELECT Robots.* FROM Robots WHERE id NOT IN (1, 2, 3)'
$builder->from('Robots')
    ->notInWhere('id', array(1, 2, 3));
```

```
// 'SELECT Robots.* FROM Robots WHERE name LIKE '%Art%'
$builder->from('Robots')
    ->where('name LIKE :name:', array('name' => '%', . $name . '%'));

// 'SELECT r.* FROM Store\Robots WHERE r.name LIKE '%Art%'
$builder->from(['r' => 'Store\Robots'])
    ->where('r.name LIKE :name:', array('name' => '%', . $name . '%'));
```

## Связанные параметры

В Query Builder можно устанавливать связанные параметры, указывать их можно непосредственно в запросе, либо в момент выполнения:

```
<?php

// Указываем параметры в формирующих участках
$robots = $this->modelsManager->createBuilder()
    ->from('Robots')
    ->where('name = :name:', array('name' => $name))
    ->andWhere('type = :type:', array('type' => $type))
    ->getQuery()
    ->execute();

// Указываем параметры при выполнении запроса
$robots = $this->modelsManager->createBuilder()
    ->from('Robots')
    ->where('name = :name:')
    ->andWhere('type = :type:')
    ->getQuery()
    ->execute(array('name' => $name, 'type' => $type));
```

### 2.12.8 Запрет на константы в PHQL

Константы можно отключить в PHQL, это означает, что напрямую строки, числа или булевы значения использовать в PHQL будет нельзя. Если PHQL запросы создаются со встраиванием внешних данных с помощью констант, то это может открыть приложение для потенциальных SQL-инъекций:

```
<?php

$login = 'voltron';
$phql = "SELECT * FROM Models\Users WHERE login = '$login'";
$result = $manager->executeQuery($phql);
```

Если значение \$login заменить на ‘ OR ‘ = ‘, то получим следующий PHQL:

```
<?php

"SELECT * FROM Models\Users WHERE login = '' OR '' = ''"
```

Что всегда имеет место быть, независимо от того, что логин хранится в базе данных.

Если константы запрещены, строки могут быть использованы как часть PHQL запроса, таким образом будет брошено исключение, заставляющее разработчика использовать связанные параметры. Этот же запрос можно записать в безопасном виде вот так:

```
<?php
```

```
$phql = "SELECT Robots.* FROM Robots WHERE Robots.name = :name:";  
$result = $manager->executeQuery($phql, array('name' => $name));
```

Запретить константы можно следующим способом:

```
<?php
```

```
Phalcon\Mvc\Model::setup(array('phqlLiterals' => false));
```

Связанные параметры можно использовать, даже если константы разрешены. Запрет на них является еще одним безопасным решением, которое разработчик может использовать в web-приложениях.

## 2.12.9 Экранирование зарезервированных слов

У PHQL есть несколько зарезервированных слов, и если вы хотите использовать какое-то из них в качестве атрибутов или названий моделей, то вам придётся их экранировать с помощью '[' и ']':

```
<?php
```

```
$phql = "SELECT * FROM [Update]";  
$result = $manager->executeQuery($phql);  
  
$phql = "SELECT id, [Like] FROM Posts";  
$result = $manager->executeQuery($phql);
```

Эти разделители будут динамически преобразованы в валидные разделители той СУБД, которая используется приложением в текущий момент.

## 2.12.10 Жизненный цикл PHQL

Будучи высокоуровневым языком, PHQL даёт разработчикам возможность персонализировать и настраивать различные аспекты под свои нужды. Ниже представлен жизненный цикл исполнения каждого PHQL-оператора:

- PHQL разбирает и преобразует в промежуточное представление, независящее от текущей СУБД
- Это промежуточное представление преобразуется в валидный SQL, соответствующий СУБД, связанной с моделью
- Все параметры и сформированный PHQL запрос кэшируется в памяти. Повторные выполнения этого же запроса производятся в разы быстрее

## 2.12.11 Использование чистого SQL

СУБД могут предлагать свои специфические SQL-расширения, не поддерживаемые PHQL, в этом случае можно использовать чистый SQL:

```
<?php
```

```
use Phalcon\Mvc\Model\Resultset\Simple as Resultset;  
  
class Robots extends Phalcon\Mvc\Model  
{
```

```

public static function findByCreateInterval()
{
    // Выражение на чистом SQL
    $sql = "SELECT * FROM robots WHERE id > 0";

    // Модель
    $robot = new Robots();

    // Выполнение запроса
    return new Resultset(null, $robot, $robot->getReadConnection()->query($sql));
}
}

```

Если чистые SQL-запросы являются общими для вашего приложения, то в модель можно добавить универсальный метод:

```

<?php

use Phalcon\Mvc\Model\Resultset\Simple as Resultset;

class Robots extends Phalcon\Mvc\Model
{
    public static function findByRawSql($conditions, $params=null)
    {
        // Выражение на чистом SQL
        $sql = "SELECT * FROM robots WHERE $conditions";

        // Модель
        $robot = new Robots();

        // Выполнение запроса
        return new Resultset(null, $robot, $robot->getReadConnection()->query($sql, $params));
    }
}

```

Определённый выше метод findByRawSql может быть использован следующим образом:

```

<?php

$robots = Robots::findByRawSql('id > ?', array(10));

```

## 2.12.12 Поиск и исправление проблем

Имейте в виду следующие моменты, когда используете PHQL:

- Классы регистрозависимы, если класс не определён так, как он определён, то это может привести к неожиданному поведению.
- Чтобы успешно связывать (bind) параметры, в соединении должна быть определена правильная кодировка.
- Классы, для которых заданы алиасы не заменяются классами с неймспейсами, поскольку это происходит только в PHP коде, а не внутри строк.

## 2.13 Кэширование в ORM

Каждое приложение уникально: у нас могут быть модели с часто изменяемыми данными, так и модели с данными, которые редко изменяют свои значения. Обращение к базе данных часто является одним из наиболее распространенных узких мест в плане производительности приложения. Это связано со сложными процессами подключения/коммуникации, которые PHP должен выполнять при каждом запросе к базе данных для получения требуемых данных. Поэтому, если мы хотим добиться хорошей производительности, мы должны добавить несколько слоев кэширования, когда приложение в этом нуждается.

В этой главе рассматриваются места, где можно реализовать кэширование для повышения производительности. Фреймворк дает вам инструмент для реализации кэша, в тех местах где вам нужно в соответствии с архитектурой приложения.

### 2.13.1 Кэширование наборов данных

Имеется методика позволяющая избежать постоянного обращения к базе данных, это кэширование редко изменяемых наборов данных, используя систему с более быстрым доступом (обычно это память).

Когда *Phalcon|Mvc|Model* требуется сервис для кэша наборов данных, он будет запрашивать у контейнера зависимостей этот сервис с именем «modelsCache».

Phalcon предоставляет компонент *cache* для кэширования любых данных, мы объясним как интегрировать его с моделями. Во-первых, вы должны зарегистрировать его в качестве сервиса в контейнере зависимостей:

```
<?php
```

```
// Регистрация сервиса кэша моделей
$di->set('modelsCache', function() {

    //По умолчанию данные кэша хранятся один день
    $frontCache = new \Phalcon\Cache\Frontend\Data(array(
        "lifetime" => 86400
    ));

    // Настройки соединения с memcached
    $cache = new \Phalcon\Cache\Backend\Memcache($frontCache, array(
        "host" => "localhost",
        "port" => "11211"
    ));

    return $cache;
});
```

Вы имеете полный контроль в создании и настройке кэша перед его использованием путем регистрации сервиса в качестве анонимной функции. После того, как настройка кэша правильно определена, можно кэшировать наборы данных:

```
<?php
```

```
// Получить продукта без кэширования
$products = Products::find();

// Используем кэширование наборов данных. Кэш остается в памяти в течении 1 часа (3600 секунд).
$products = Products::find(array(
    "cache" => array("key" => "my-cache")
```

```
));

// Кэш набора данных хранится всего 5 минут
$products = Products::find(array(
    "cache" => array("key" => "my-cache", "lifetime" => 300)
));

// Использование пользовательского кэша
$products = Products::find(array("cache" => $myCache));
```

Кэш может быть также применен к набору данных, генерируемых с помощью отношений:

```
<?php

// Запрос некоторого сообщения
$post = Post::findFirst();

// Получаем комментарии, относящиеся к сообщению, и кэшируем их
$comments = $post->getComments(array(
    "cache" => array("key" => "my-key")
));

// Получаем комментарии относящиеся к сообщению и устанавливаем срок их хранения
$comments = $post->getComments(array(
    "cache" => array("key" => "my-key", "lifetime" => 3600)
));
```

Когда кэшируемые наборы данных должны быть признаны недействительными, вы можете просто удалить их из кэша с использованием ранее указанного ключа.

Обратите внимание, что не все наборы данных должны быть в кэше. Данные, которые меняют свои значения очень часто не следует кэшировать, так как они становятся не действительными очень быстро, и кэширование в этом случае отрицательно влияет на производительность приложения. Кроме того, большие наборы данных, которые не часто меняют свои значения, могут располагаться в кэше, но для реализации этой идеи необходимо оценить имеющиеся механизмы кэширования и влияния на производительность, так как это не всегда будет способствовать увеличению производительности приложения.

## 2.13.2 Переопределение find/findFirst

Как показано выше, эти методы доступны в моделях, которые наследуют *Phalcon\ Mvc\ Model*:

```
<?php

class Robots extends Phalcon\ Mvc\ Model
{

    public static function find($parameters=null)
    {
        return parent::find($parameters);
    }

    public static function findFirst($parameters=null)
    {
        return parent::findFirst($parameters);
    }
}
```

```
}
```

Сделав это, вы будете перехватывать все вызовы этих методов, таким образом, вы можете добавить кэширующий слой или запускать запросы к базе данных, если кэша нет. Например, очень простой реализацией кэша является использование статического свойства, чтобы избежать того, что запись будет запрашиваться несколько раз в одной и том же запросе:

```
<?php

class Robots extends Phalcon\Mvc\Model
{

    protected static $_cache = array();

    /**
     * Реализация метода, который возвращает
     * строковый ключ на основе параметров запроса
     */
    protected static function _createKey($parameters)
    {
        $uniqueKey = array();
        foreach ($parameters as $key => $value) {
            if (is_scalar($value)) {
                $uniqueKey[] = $key . ':' . $value;
            } else {
                if (is_array($value)) {
                    $uniqueKey[] = $key . ':' . self::_createKey($value) . '';
                }
            }
        }
        return join(',', $uniqueKey);
    }

    public static function find($parameters=null)
    {

        // Создание уникального ключа на основе параметров
        $key = self::_createKey($parameters);

        if (!isset(self::$_cache[$key])) {
            //Store the result in the memory cache
            self::$_cache[$key] = parent::find($parameters);
        }

        // Вернуть результат в кэше
        return self::$_cache[$key];
    }

    public static function findFirst($parameters=null)
    {
        // ...
    }
}
```

Доступ к базе данных в несколько раз медленнее, чем вычисление ключа кэша, вы свободны в реализации стратегии генерации ключа, которая лучше подходит для ваших задач. Следует отметить, что хороший ключ позволяет избежать конфликтов, насколько это возможно, это означает, что разные

ключи возвращают unrelated records to the find parameters.

В приведенном выше примере мы использовали кэш в памяти, он полезен в качестве первого уровня кэша. Как только у нас есть кэш в памяти, мы можем реализовать слой кэша второго уровня с помощью APC / XCache или базы данных NoSQL:

```
<?php

public static function find($parameters=null)
{

    // Создание уникального ключа на основе параметров
    $key = self::_createKey($parameters);

    if (!isset(self::$_cache[$key])) {

        //Мы используем APC как кэш второго уровня
        if (apc_exists($key)) {

            $data = apc_fetch($key);

            //Сохраним результат в кэш памяти
            self::$_cache[$key] = $data;

            return $data;
        }

        //Если нет кэша в памяти или в APC
        $data = parent::find($parameters);

        //Сохраним результат в кэш памяти
        self::$_cache[$key] = $data;

        //Сохраним результат в APC
        apc_store($key, $data);

        return $data;
    }

    //Вернуть результат в кэше
    return self::$_cache[$key];
}
```

Это дает вам полный контроль над тем, как кэши должны быть реализованы для каждой модели, эта стратегия может быть общей для нескольких моделей, которую можно вынести в отдельный базовый класс для всех подобных классов:

```
<?php

class CacheableModel extends Phalcon\Mvc\Model
{

    protected static function _createKey($parameters)
    {
        // .. create a cache key based on the parameters
    }

    public static function find($parameters=null)
    {
```

```
    ... custom caching strategy
}

public static function findFirst($parameters=null)
{
    ... custom caching strategy
}
}
```

Затем используйте этот класс в качестве базового класса для каждой модели ‘Cacheable’:

```
<?php

class Robots extends CacheableModel
{
```

### 2.13.3 Форсирование кэша

Ранее мы видели, как Phalcon\Mvc\Model имеет встроенную интеграцию с компонентом кэширования, предоставленного фреймворком. Чтобы сделать запись/результатирующую набор кэшируемым, мы передаем ключ ‘cache’ в массиве параметров:

```
<?php

// Кэшируем результатирующий набор всего на 5 минут
$products = Products::find(array(
    "cache" => array("key" => "my-cache", "lifetime" => 300)
));
```

Это дает нам свободу для кэширования конкретных запросов, поэтому если мы хотим кэшировать глобально все запросы, выполняемые моделью, мы можем переопределить метод find/findFirst, чтобы заставить кэшировать каждый запрос.

```
<?php

class Robots extends Phalcon\Mvc\Model
{

    protected static function _createKey($parameters)
    {
        // .. создаем ключ кэша на основе параметров
    }

    public static function find($parameters=null)
    {

        // Преобразование параметров в массив
        if (!is_array($parameters)) {
            $parameters = array($parameters);
        }

        // Проверяем, что ключ кэша не был передан
        // и создаем параметры кэша
        if (!isset($parameters['cache'])) {
            $parameters['cache'] = array(
```

```

        "key" => self::$_createKey($parameters),
        "lifetime" => 300
    );
}

return parent::find($parameters);
}

public static function findFirst($parameters=null)
{
    //...
}

```

## 2.13.4 Кэширование PHQL запросов

Все запросы в ORM, независимо от того, насколько высокоуровневый синтаксис мы использовали для их создания, обрабатываются внутри с помощью PHQL. Этот язык дает гораздо больше свободы для создания запросов всех видов. Конечно, эти запросы могут кэшироваться:

```

<?php

$phql = "SELECT * FROM Cars WHERE name = :name:";

$query = $this->modelsManager->createQuery($phql);

$query->setCache(array(
    "key" => "cars-by-name",
    "lifetime" => 300
));

$cars = $query->execute(array(
    'name' => 'Audi'
));

```

Если вы не хотите использовать неявный кэш, просто сохраните результирующий набор в предпочтительный для вас серверный кэш:

```

<?php

$phql = "SELECT * FROM Cars WHERE name = :name:";

$cars = $this->modelsManager->executeQuery($phql, array(
    'name' => 'Audi'
));

apc_store('my-cars', $cars);

```

## 2.13.5 Многократное использование связанных записей

Некоторые модели могут иметь связи с другими моделями. Это позволяет нам легко проверить записи, которые относятся к экземплярам в памяти:

```
<?php

// Получаем некоторый счет
$invoice = Invoices::findFirst();

// Получаем клиента связанного со счетом
$customer = $invoice->customer;

// Выводим его/ее имя
echo $customer->name, "\n";
```

Этот пример очень простой, клиент получает запрос и который может быть использован при необходимости, например, чтобы показать свое имя. Это также касается случаев если мы извлекаем наборы счетов, чтобы показать клиентам, которые являются владельцами этих счетов:

```
<?php

// Получаем набор счетов
// SELECT * FROM invoices
foreach (Invoices::find() as $invoice) {

    // Получаем клиента связанного с заказом
    // SELECT * FROM customers WHERE id = ?
    $customer = $invoice->customer;

    // Выводим его/ее имя
    echo $customer->name, "\n";
}
```

Клиент может иметь один или несколько счетов, это означает, что клиент может быть вызван вызван более одного раза. Чтобы избежать этого, мы можем отметить связь как многоразовую , таким образом, мы говорим ORM автоматически использовать прошлые записи вместо того, чтобы вновь и вновь выполнять один и тот же запросы:

```
<?php

class Invoices extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->belongsTo("customers_id", "Customer", "id", array(
            'reusable' => true
        ));
    }
}
```

Этот кэш работает только в памяти, это означает, что кэшированные данные предоставляются, когда запрос уже был выполнен. Вы можете добавить более сложные кэш для этого сценария, переопределив менеджер модели:

```
<?php

class CustomModelsManager extends \Phalcon\Mvc\Model\Manager
{

    /**
     * Возвращает многократно используемый объект из кэша

```

```

/*
 * @param string $modelName
 * @param string $key
 * @return object
 */
public function getReusableRecords($modelName, $key){

    // Если модель Products использует кэш APC
    if ($modelName == 'Products'){
        return apc_fetch($key);
    }

    // Для остальных, использовать кэш памяти
    return parent::getReusableRecords($modelName, $key);
}

/**
 * Сохраняет повторно используемый запись в кэше
 *
 * @param string $modelName
 * @param string $key
 * @param mixed $records
 */
public function setReusableRecords($modelName, $key, $records){

    // Если модель Products использует кэш APC
    if ($modelName == 'Products'){
        apc_store($key, $records);
        return;
    }

    // Для остальных, использовать кэш памяти
    parent::setReusableRecords($modelName, $key, $records);
}
}

```

Не забудьте зарегистрировать свой менеджер моделей в DI:

```
<?php

$di->setShared('modelsManager', function() {
    return new CustomModelsManager();
});
```

## 2.13.6 Кэширование связанных записей

Когда запрашиваются связанные записи, внутри ORM строится соответствующие состояния, и передаются необходимые записи с помощью Find / FindFirst в целевую модель в соответствии со следующей таблицей:

| Тип        | Описание  | Вызываемый метод |
|------------|---|------------------|
| Belongs-To | Возвращает непосредственно экземпляр модели взаимосвязанной записи  | findFirst        |
| Has-One    | Возвращает непосредственно экземпляр модели взаимосвязанной записи  | findFirst        |
| Has-Many   | Возвращает коллекцию экземпляров модели которые ссылаются на модель | find             |

Это означает, что когда вы получаете связанные записи, вы можете изменить способ получения данных путем реализации соответствующего метода:

```
<?php

// Получаем счет
$invoice = Invoices::findFirst();

// Получаем владельца счета
$customer = $invoice->customer; // Invoices::findFirst('...');

// То же самое
$customer = $invoice->getCustomer(); // Invoices::findFirst('...');
```

Соответственно, мы могли бы заменить метод FindFirst в модели счетов и осуществлять кэширование наиболее подходящим способом:

```
<?php

class Invoices extends Phalcon\Mvc\Model
{

    public static function findFirst($parameters=null)
    {
        //.. здесь реализуем кэширование данных
    }
}
```

### 2.13.7 Рекурсивное кэширование связанных записей

В этом сценарии мы предполагаем, что каждый раз когда мы запрашиваем набор данных, мы также получить все связанные записи для данного набора. Если мы будем хранить записи найденные вместе с их связанными сущностями, возможно, мы сможем немного уменьшить накладные расходы для получения всех сущностей:

```
<?php

class Invoices extends Phalcon\Mvc\Model
{

    protected static function _createKey($parameters)
    {
        // .. создаем ключ кэша на основе параметров
    }

    protected static function _getCache($key)
    {
        // .. возвращаем данные из кэша
    }
}
```

```

}

protected static function _setCache($key)
{
    // .. сохраняет данные в кэше
}

public static function find($parameters=null)
{
    // Создать уникальный ключ
    $key = self::_createKey($parameters);

    // Проверяем наличие данных в кэше
    $results = self::_getCache($key);

    // Полученные данные должны быть объектом
    if (is_object($results)) {
        return $results;
    }

    $results = array();

    $invoices = parent::find($parameters);
    foreach ($invoices as $invoice) {

        // Получение соответствующего клиента
        $customer = $invoice->customer;

        // Помещаем его в запись
        $invoice->customer = $customer;

        $results[] = $invoice;
    }

    // Сохраняем счета и их клиентов в кэше
    self::_setCache($key, $results);

    return $results;
}

public function initialize()
{
    // .. добавляем связи и инициализируем другие вещи
}
}

```

Получение из кэша счетов уже содержащих данные о клиентах выполняется всего за одно действие, что снижает общую нагрузку на данную операцию. Следует отметить, что этот процесс можно также проводить с PHQL с помощью следующего альтернативного решения:

```

<?php

class Invoices extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        // .. добавляем связи и инициализируем другие вещи
    }
}

```

```
}

protected static function _createKey($conditions, $params)
{
    // .. создаем ключ кэша на основе параметров
}

public function getInvoicesCustomers($conditions, $params=null)
{
    $phql = "SELECT Invoices.*, Customers.*  
FROM Invoices JOIN Customers WHERE " . $conditions;

    $query = $this->getModelsManager()->executeQuery($phql);

    $query->setCache(array(
        "key" => self::_createKey($conditions, $params),
        "lifetime" => 300
    ));

    return $query->execute($params);
}

}


```

## 2.13.8 Кэширование на основе условий

В этом случае, кэш реализуется в соответствии с текущими полученными условиями. В соответствии с областью, куда попадает первичный ключ, выбирается соответствующий способ кэширования.

| Значение      | Способ кэширования |
|---------------|--------------------|
| 1 - 10000     | mongo1             |
| 10000 - 20000 | mongo2             |
| > 20000       | mongo3             |

Самый простой способ это добавление статического метода к модели, который выбирает правильный кэш для использования:

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public static function queryCache($initial, $final)
    {
        if ($initial >= 1 && $final < 10000) {
            return self::find(array(
                'id >= ' . $initial . ' AND id <= ' . $final,
                'cache' => array('service' => 'mongo1')
            ));
        }
        if ($initial >= 10000 && $final <= 20000) {
            return self::find(array(
                'id >= ' . $initial . ' AND id <= ' . $final,
                'cache' => array('service' => 'mongo2')
            ));
        }
        if ($initial > 20000) {


```

```

        return self::find(array(
            'id >= ' . $initial,
            'cache' => array('service' => 'mongo3')
        ));
    }
}

}

```

Такой подход решает проблему, однако, если мы хотим добавить другие параметры, такие как сортировка или условия, мы должны были бы создать более сложный метод. Кроме того, этот метод не работает, если данные получаются с использованием связанных записей или find/FindFirst:

```

<?php

$robots = Robots::find('id < 1000');
$robots = Robots::find('id > 100 AND type = "A"');
$robots = Robots::find('(id > 100 AND type = "A") AND id < 2000');

$robots = Robots::find(array(
    '(id > ?0 AND type = "A") AND id < ?1',
    'bind' => array(100, 2000),
    'order' => 'type'
));

```

Для достижения этой цели мы должны перехватить промежуточное представление (IR), порожденную PHQL анализатором и таким образом получить возможность настроить способы кэширования:

Для начала, необходимо реализовать пользовательский конструктор запросов, в котором мы сможем генерировать полностью настраиваемые запросы к базе данных:

```

<?php

class CustomQueryBuilder extends Phalcon\Mvc\Model\Query\Builder
{

    public function getQuery()
    {
        $query = new CustomQuery($this->getPhql());
        $query->setDI($this->getDI());
        return $query;
    }

}

```

Вместо того, чтобы непосредственно возвращать Phalcon\ Mvc\ Model\ Query, наш конструктор возвращает экземпляр класса CustomQuery, этот класс выглядит следующим образом:

```

<?php

class CustomQuery extends Phalcon\Mvc\Model\Query
{

    /**
     * The execute method is overridden
     */
    public function execute($params=null, $types=null)
    {
        // Разбор промежуточных представлений для SELECT

```

```
$ir = $this->parse();

// Проверяем, что наш запрос имеет условия
if (isset($ir['where'])) {

    // Поля в условии могут иметь любой порядок
    // Нам нужно рекурсивно проверить дерево условий,
    // чтобы найти информацию, которую мы ищем
    $visitor = new CustomNodeVisitor();

    // Рекурсивно просматриваем узлы
    $visitor->visit($ir['where']);

    $initial = $visitor->getInitial();
    $final = $visitor->getFinal();

    // Выбираем кэш в зависимости от диапазона
    //...

    // Проверяем, что кэш имеет данные
    //...
}

// Выполняем запрос
$result = $this->_executeSelect($ir, $params, $types);

// Сохраняем результат в кэш

//...

return $result;
}
```

Реализация помощника (`CustomNodeVisitor`), который рекурсивно проверяет условия на наличие полей, которые передают диапазон возможных значений, который будет использоваться при кэшировании:

```
<?php

class CustomNodeVisitor
{
    protected $_initial = 0;

    protected $_final = 25000;

    public function visit($node)
    {
        switch ($node['type']) {
            case 'binary-op':
                $left = $this->visit($node['left']);
                $right = $this->visit($node['right']);
                if (!$left || !$right) {
                    return false;
                }
        }
    }
}
```

```

        if ($left=='id') {
            if ($node['op'] == '>') {
                $this->_initial = $right;
            }
            if ($node['op'] == '=') {
                $this->_initial = $right;
            }
            if ($node['op'] == '>=') {
                $this->_initial = $right;
            }
            if ($node['op'] == '<') {
                $this->_final = $right;
            }
            if ($node['op'] == '<=') {
                $this->_final = $right;
            }
        }
    }
    break;

case 'qualified':
    if ($node['name'] == 'id') {
        return 'id';
    }
    break;

case 'literal':
    return $node['value'];

default:
    return false;
}

public function getInitial()
{
    return $this->_initial;
}

public function getFinal()
{
    return $this->_final;
}
}

```

Наконец, мы можем заменить поисковый метод в модели Robots и использовать пользовательские классы, которые мы создали:

```

<?php

class Robots extends Phalcon\Mvc\Model
{
    public static function find($parameters=null)
    {

        if (!is_array($parameters)) {
            $parameters = array($parameters);
        }
    }
}

```

```
$builder = new CustomQueryBuilder($parameters);
$builder->from(get_called_class());

if (isset($parameters['bind'])) {
    return $builder->getQuery()->execute($parameters['bind']);
} else {
    return $builder->getQuery()->execute();
}

}

}
```

### 2.13.9 Caching of PHQL planning

As well as most moderns database systems PHQL internally caches the execution plan, if the same statement is executed several times PHQL reuses the previously generated plan improving performance, for a developer to take better advantage of this is highly recommended build all your SQL statements passing variable parameters as bound parameters:

```
<?php

for ($i = 1; $i <= 10; $i++) {

    $phql = "SELECT * FROM Store\Robots WHERE id = " . $i;
    $robots = $this->modelsManager->executeQuery($phql);

    //...
}
```

In the above example, ten plans were generated increasing the memory usage and processing in the application. Rewriting the code to take advantage of bound parameters reduces the processing by both ORM and database system:

```
<?php

$phql = "SELECT * FROM Store\Robots WHERE id = ?0";

for ($i = 1; $i <= 10; $i++) {

    $robots = $this->modelsManager->executeQuery($phql, array($i));

    //...
}
```

Performance can be also improved reusing the PHQL query:

```
<?php

$phql = "SELECT * FROM Store\Robots WHERE id = ?0";
$query = $this->modelsManager->createQuery($phql);

for ($i = 1; $i <= 10; $i++) {

    $robots = $query->execute($phql, array($i));

    //...
}
```

Execution plans for queries involving [prepared statements](#) are also cached by most database systems reducing the overall execution time, also protecting your application against [SQL Injections](#).

## 2.14 ODM (Object-Document Mapper)

In addition to its ability to [map tables](#) in relational databases, Phalcon can map documents from NoSQL databases. The ODM offers a CRUD functionality, events, validations among other services.

Due to the absence of SQL queries and planners, NoSQL databases can see real improvements in performance using the Phalcon approach. Additionally, there are no SQL building reducing the possibility of SQL injections.

The following NoSQL databases are supported:

| Name    | Description  |
|---------|--|
| MongoDB | MongoDB is a scalable, high-performance, open source NoSQL database. |

### 2.14.1 Creating Models

A model is a class that extends from [Phalcon\ Mvc\ Collection](#). It must be placed in the models directory. A model file must contain a single class; its class name should be in camel case notation:

```
<?php

class Robots extends \Phalcon\ Mvc\ Collection
{

}
```

If you're using PHP 5.4/5.5 is recommended declare each column that makes part of the model in order to save memory and reduce the memory allocation.

By default model "Robots" will refer to the collection "robots". If you want to manually specify another name for the mapping collection, you can use the getSource() method:

```
<?php

class Robots extends \Phalcon\ Mvc\ Collection
{
    public function getSource()
    {
        return "the_robots";
    }
}
```

### 2.14.2 Understanding Documents To Objects

Every instance of a model represents a document in the collection. You can easily access collection data by reading object properties. For example, for a collection "robots" with the documents:

```
$ mongo test
MongoDB shell version: 1.8.2
connecting to: test
> db.robots.find()
{ "_id" : ObjectId("508735512d42b8c3d15ec4e1"), "name" : "Astro Boy", "year" : 1952,
```

```
"type" : "mechanical" }  
{ "_id" : ObjectId("5087358f2d42b8c3d15ec4e2"), "name" : "Bender", "year" : 1999,  
    "type" : "mechanical" }  
{ "_id" : ObjectId("508735d32d42b8c3d15ec4e3"), "name" : "Wall-E", "year" : 2008 }  
>
```

### 2.14.3 Models in Namespaces

Namespaces can be used to avoid class name collision. In this case it is necessary to indicate the name of the related collection using getSource:

```
<?php  
  
namespace Store\Toys;  
  
class Robots extends \Phalcon\Mvc\Collection  
{  
  
    public function getSource()  
    {  
        return "robots";  
    }  
  
}
```

You could find a certain document by its id and then print its name:

```
<?php  
  
// Find record with _id = "5087358f2d42b8c3d15ec4e2"  
$robot = Robots::findId("5087358f2d42b8c3d15ec4e2");  
  
// Prints "Bender"  
echo $robot->name;
```

Once the record is in memory, you can make modifications to its data and then save changes:

```
<?php  
  
$robot = Robots::findFirst(  
    array('name' => 'Astroy Boy')  
>);  
$robot->name = "Voltron";  
$robot->save();
```

### 2.14.4 Setting a Connection

Connections are retrieved from the services container. By default, Phalcon tries to find the connection in a service called “mongo”:

```
<?php  
  
// Simple database connection to localhost  
$di->set('mongo', function() {  
    $mongo = new Mongo();  
    return $mongo->selectDb("store");
```

```

}, true);

// Connecting to a domain socket, falling back to localhost connection
$di->set('mongo', function() {
    $mongo = new Mongo("mongodb://tmp/mongodb-27017.sock,localhost:27017");
    return $mongo->selectDb("store");
}, true);

```

## 2.14.5 Finding Documents

As *Phalcon\ Mvc\Collection* relies on the Mongo PHP extension you have the same facilities to query documents and convert them transparently to model instances:

```

<?php

// How many robots are there?
$robots = Robots::find();
echo "There are ", count($robots), "\n";

// How many mechanical robots are there?
$robots = Robots::find(array(
    array("type" => "mechanical")
));
echo "There are ", count($robots), "\n";

// Get and print mechanical robots ordered by name upward
$robots = Robots::find(array(
    array("type" => "mechanical"),
    "sort" => array("name" => 1)
));

foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

// Get first 100 mechanical robots ordered by name
$robots = Robots::find(array(
    array("type" => "mechanical"),
    "sort" => array("name" => 1),
    "limit" => 100
));

foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

```

You could also use the `findFirst()` method to get only the first record matching the given criteria:

```

<?php

// What's the first robot in robots collection?
$robot = Robots::findFirst();
echo "The robot name is ", $robot->name, "\n";

// What's the first mechanical robot in robots collection?
$robot = Robots::findFirst(array(
    array("type" => "mechanical")
)

```

```
));
echo "The first mechanical robot name is ", $robot->name, "\n";
```

Both find() and findFirst() methods accept an associative array specifying the search criteria:

```
<?php

// First robot where type = "mechanical" and year = "1999"
$robot = Robots::findFirst(array(
    "type" => "mechanical",
    "year" => "1999"
));

// All virtual robots ordered by name downward
$robots = Robots::find(array(
    "conditions" => array("type" => "virtual"),
    "sort"         => array("name" => -1)
));
```

The available query options are:

| Parameter  | Description  | Example                                       |
|------------|--|---|
| conditions | Search conditions for the find operation. Is used to extract only those records that fulfill a specified criterion. By default Phalcon_model assumes the first parameter are the conditions. | “conditions” => array('\$gt' => 1990)         |
| fields     | Returns specific columns instead of the full fields in the collection. When using this option an incomplete object is returned   | “conditions” => array('\$gt' => 1990)         |
| sort       | It's used to sort the resultset. Use one or more fields as each element in the array, 1 means ordering upwards, -1 downward  | “order” => array("name" => -1, "statys" => 1) |
| limit      | Limit the results of the query to results to certain range   | “limit” => 10                                 |
| skip       | Skip a number of results   | “skip” => 50                                  |

If you have experience with SQL databases, you may want to check the [SQL to Mongo Mapping Chart](#).

## 2.14.6 Aggregations

A model can return calculations using aggregation framework provided by Mongo. The aggregated values are calculate without having to use MapReduce. With this option is easy perform tasks such as totaling or averaging field values:

```
<?php

$data = Article::aggregate(array(
    array(
        '$project' => array('category' => 1)
    ),
    array(
        '$group' => array(
            '_id' => array('category' => '$category'),
            'id'  => array('$max' => '$_id')
        )
    )
));
```

## 2.14.7 Creating Updating/Records

The method `Phalcon\Mvc\Collection::save()` allows you to create/update documents according to whether they already exist in the collection associated with a model. The ‘`save`’ method is called internally by the create and update methods of `Phalcon\ Mvc\ Collection`.

Also the method executes associated validators and events that are defined in the model:

```
<?php

$robot      = new Robots();
$robot->type = "mechanical";
$robot->name = "Astro Boy";
$robot->year = 1952;
if ($robot->save() == false) {
    echo "Umh, We can't store robots right now: \n";
    foreach ($robot->getMessages() as $message) {
        echo $message, "\n";
    }
} else {
    echo "Great, a new robot was saved successfully!";
}
```

The “`_id`” property is automatically updated with the `MongoId` object created by the driver:

```
<?php

$robot->save();
echo "The generated id is: ", $robot->getId();
```

## Validation Messages

`Phalcon\ Mvc\ Collection` has a messaging subsystem that provides a flexible way to output or store the validation messages generated during the insert/update processes.

Each message consists of an instance of the class `Phalcon\ Mvc\ Model\ Message`. The set of messages generated can be retrieved with the method `getMessages()`. Each message provides extended information like the field name that generated the message or the message type:

```
<?php

if ($robot->save() == false) {
    foreach ($robot->getMessages() as $message) {
        echo "Message: ", $message->getMessage();
        echo "Field: ", $message->getField();
        echo "Type: ", $message->getType();
    }
}
```

## Validation Events and Events Manager

Models allow you to implement events that will be thrown when performing an insert or update. They help define business rules for a certain model. The following are the events supported by `Phalcon\ Mvc\ Collection` and their order of execution:

| Operation                         | Name  | Can stop operation? | Explanation  |
|-----------------------------------|---|---------------------|--|
| Inserting/UpdatingValidation      | beforeValidationOnCreate                      | YES                 | Is executed before the validation process and the final insert/update to the database                              |
| Inserting                         | beforeValidationOnCreate                      | NO                  | Is executed before the validation process only when an insertion operation is being made                           |
| Updating                          | beforeValidationOnUpdate                      | NO                  | Is executed before the fields are validated for not nulls or foreign keys when an updating operation is being made |
| Inserting/UpdatingValidationFails | beforeValidationOnCreate<br>(already stopped) | YES                 | Is executed before the validation process only when an insertion operation is being made                           |
| Inserting                         | afterValidationOnCreate                       | YES                 | Is executed after the validation process when an insertion operation is being made                                 |
| Updating                          | afterValidationOnUpdate                       | YES                 | Is executed after the validation process when an updating operation is being made                                  |
| Inserting/UpdatingValidation      | afterValidationOnSave                         | YES                 | Is executed after the validation process   |
| Inserting/UpdatingSave            |   | YES                 | Runs before the required operation over the database system  |
| Updating                          | beforeUpdate                                  | YES                 | Runs before the required operation over the database system only when an updating operation is being made          |
| Inserting                         | beforeCreate                                  | YES                 | Runs before the required operation over the database system only when an inserting operation is being made         |
| Updating                          | afterUpdate                                   | NO                  | Runs after the required operation over the database system only when an updating operation is being made           |
| Inserting                         | afterCreate                                   | NO                  | Runs after the required operation over the database system only when an inserting operation is being made          |
| Inserting/UpdatingSave            |   | NO                  | Runs after the required operation over the database system   |

To make a model to react to an event, we must implement a method with the same name of the event:

```
<?php

class Robots extends \Phalcon\Mvc\Collection
{

    public function beforeValidationOnCreate()
    {
        echo "This is executed before creating a Robot!";
    }

}
```

Events can be useful to assign values before performing an operation, for example:

```
<?php

class Products extends \Phalcon\Mvc\Collection
{

    public function beforeCreate()
    {
        // Set the creation date
        $this->created_at = date('Y-m-d H:i:s');
    }

    public function beforeUpdate()
    {
```

```

    // Set the modification date
    $this->modified_in = date('Y-m-d H:i:s');
}

}

```

Additionally, this component is integrated with *Phalcon|Events|Manager*, this means we can create listeners that run when an event is triggered.

```

<?php

$eventsManager = new Phalcon\Events\Manager();

//Attach an anonymous function as a listener for "model" events
$eventsManager->attach('collection', function($event, $robot) {
    if ($event->getType() == 'beforeSave') {
        if ($robot->name == 'Scooby Doo') {
            echo "Scooby Doo isn't a robot!";
            return false;
        }
    }
    return true;
});

$robot = new Robots();
$robot->setEventsManager($eventsManager);
$robot->name = 'Scooby Doo';
$robot->year = 1969;
$robot->save();

```

In the example given above the EventsManager only acted as a bridge between an object and a listener (the anonymous function). If we want all objects created in our application use the same EventsManager, then we need to assign this to the Models Manager:

```

<?php

//Registering the collectionManager service
$di->set('collectionManager', function() {

    $eventsManager = new Phalcon\Events\Manager();

    // Attach an anonymous function as a listener for "model" events
    $eventsManager->attach('collection', function($event, $model) {
        if (get_class($model) == 'Robots') {
            if ($event->getType() == 'beforeSave') {
                if ($model->name == 'Scooby Doo') {
                    echo "Scooby Doo isn't a robot!";
                    return false;
                }
            }
        }
        return true;
    });

    // Setting a default EventsManager
    $modelsManager = new Phalcon\Mvc\Collection\Manager();
    $modelsManager->setEventsManager($eventsManager);
    return $modelsManager;
});

```

```
}, true);
```

## Implementing a Business Rule

When an insert, update or delete is executed, the model verifies if there are any methods with the names of the events listed in the table above.

We recommend that validation methods are declared protected to prevent that business logic implementation from being exposed publicly.

The following example implements an event that validates the year cannot be smaller than 0 on update or insert:

```
<?php

class Robots extends \Phalcon\Mvc\Collection
{

    public function beforeSave()
    {
        if ($this->year < 0) {
            echo "Year cannot be smaller than zero!";
            return false;
        }
    }

}
```

Some events return false as an indication to stop the current operation. If an event doesn't return anything, *Phalcon\ Mvc\ Collection* will assume a true value.

## Validating Data Integrity

*Phalcon\ Mvc\ Collection* provides several events to validate data and implement business rules. The special “validation” event allows us to call built-in validators over the record. Phalcon exposes a few built-in validators that can be used at this stage of validation.

The following example shows how to use it:

```
<?php

use Phalcon\Mvc\Model\Validator\InclusionIn,
    Phalcon\Mvc\Model\Validator\Numericality;

class Robots extends \Phalcon\Mvc\Collection
{

    public function validation()
    {

        $this->validate(new InclusionIn(
            array(
                "field"  => "type",
                "message" => "Type must be: mechanical or virtual",
                "domain" => array("Mechanical", "Virtual")
            )
        ));
    }
}
```

```

        $this->validate(new Numericality(
            array(
                "field" => "price",
                "message" => "Price must be numeric"
            )
        ));

        return $this->validationHasFailed() != true;
    }

}

```

The example given above performs a validation using the built-in validator “InclusionIn”. It checks the value of the field “type” in a domain list. If the value is not included in the method, then the validator will fail and return false. The following built-in validators are available:

| Name         | Explanation  | Example                 |
|--------------|--|-------------------------|
| Email        | Validates that field contains a valid email format               | <a href="#">Example</a> |
| ExclusionIn  | Validates that a value is not within a list of possible values   | <a href="#">Example</a> |
| InclusionIn  | Validates that a value is within a list of possible values       | <a href="#">Example</a> |
| Numericality | Validates that a field has a numeric format                      | <a href="#">Example</a> |
| Regex        | Validates that the value of a field matches a regular expression | <a href="#">Example</a> |
| StringLength | Validates the length of a string                                 | <a href="#">Example</a> |

In addition to the built-in validators, you can create your own validators:

```

<?php

class UrlValidator extends \Phalcon\Mvc\Collection\Validator
{

    public function validate($model)
    {
        $field = $this->getOption('field');

        $value    = $model->$field;
        $filtered = filter_var($value, FILTER_VALIDATE_URL);
        if (!$filtered) {
            $this->appendMessage("The URL is invalid", $field, "UrlValidator");
            return false;
        }
        return true;
    }
}

```

Adding the validator to a model:

```

<?php

class Customers extends \Phalcon\Mvc\Collection
{

    public function validation()
    {
        $this->validate(new UrlValidator(array(
            "field" => "url",
        )));
    }
}

```

```
    if ($this->validationHasFailed() == true) {
        return false;
    }
}
```

The idea of creating validators is make them reusable across several models. A validator can also be as simple as:

```
<?php

class Robots extends \Phalcon\Mvc\Collection
{

    public function validation()
    {
        if ($this->type == "Old") {
            $message = new Phalcon\Mvc\Model\Message(
                "Sorry, old robots are not allowed anymore",
                "type",
                "MyType"
            );
            $this->appendMessage($message);
            return false;
        }
        return true;
    }
}
```

## 2.14.8 Deleting Records

The method `Phalcon\Mvc\Collection::delete()` allows to delete a document. You can use it as follows:

```
<?php

$robot = Robots::findFirst();
if ($robot != false) {
    if ($robot->delete() == false) {
        echo "Sorry, we can't delete the robot right now: \n";
        foreach ($robot->getMessages() as $message) {
            echo $message, "\n";
        }
    } else {
        echo "The robot was deleted successfully!";
    }
}
```

You can also delete many documents by traversing a resultset with a foreach:

```
<?php

$robots = Robots::find(array(
    array("type" => "mechanical")
));
foreach ($robots as $robot) {
    if ($robot->delete() == false) {
```

```

        echo "Sorry, we can't delete the robot right now: \n";
        foreach ($robot->getMessages() as $message) {
            echo $message, "\n";
        }
    } else {
        echo "The robot was deleted successfully!";
    }
}

```

The following events are available to define custom business rules that can be executed when a delete operation is performed:

| Operation | Name         | Can stop operation? | Explanation                              |
|-----------|--------------|---------------------|--|
| Deleting  | beforeDelete | YES                 | Runs before the delete operation is made |
| Deleting  | afterDelete  | NO                  | Runs after the delete operation was made |

## 2.14.9 Validation Failed Events

Another type of events is available when the data validation process finds any inconsistency:

| Operation                | Name              | Explanation   |
|--------------------------|-------------------|---|
| Insert or Update         | notSave           | Triggered when the insert/update operation fails for any reason |
| Insert, Delete or Update | onValidationFails | Triggered when any data manipulation operation fails            |

## 2.14.10 Implicit Ids vs. User Primary Keys

By default Phalcon\Mvc\Collection assumes that the `_id` attribute is automatically generated using MongoIDs. If a model uses custom primary keys this behavior can be overridden:

```
<?php

class Robots extends Phalcon\Mvc\Collection
{
    public function initialize()
    {
        $this->useImplicitObjectIds(false);
    }
}
```

## 2.14.11 Setting multiple databases

In Phalcon, all models can belong to the same database connection or have an individual one. Actually, when `Phalcon\ Mvc\ Collection` needs to connect to the database it requests the “mongo” service in the application’s services container. You can overwrite this service setting it in the initialize method:

```
<?php

// This service returns a mongo database at 192.168.1.100
$di->set('mongo1', function() {
    $mongo = new Mongo("mongodb://scott:nekhen@192.168.1.100");
    return $mongo->selectDb("management");
}, true);
```

```
// This service returns a mongo database at localhost
$di->set('mongo2', function() {
    $mongo = new Mongo("mongodb://localhost");
    return $mongo->selectDb("invoicing");
}, true);
```

Then, in the Initialize method, we define the connection service for the model:

```
<?php

class Robots extends \Phalcon\Mvc\Collection
{
    public function initialize()
    {
        $this->setConnectionService('mongo1');
    }

}
```

## 2.14.12 Injecting services into Models

You may be required to access the application services within a model, the following example explains how to do that:

```
<?php

class Robots extends \Phalcon\Mvc\Collection
{

    public function notSave()
    {
        // Obtain the flash service from the DI container
        $flash = $this->getDI()->getShared('flash');

        // Show validation messages
        foreach ($this->getMessages() as $message){
            $flash->error((string) $message);
        }
    }
}
```

The “notSave” event is triggered whenever a “creating” or “updating” action fails. We’re flashing the validation messages obtaining the “flash” service from the DI container. By doing this, we don’t have to print messages after each saving.

## 2.15 Использование представлений (Views)

Представления (views) — это пользовательский интерфейс вашего приложения. Чаще всего они представляют собой HTML-файлы со вставками PHP-кода, задача которого связана лишь с выводом данных. Представления управляют работой по передаче данных браузеру или любому другому средству, с помощью которого выполняются запросы к приложению.

[Phalcon\ Mvc\ View](#) и [Phalcon\ Mvc\ View\ Simple](#) отвечают за управление слоем представления в вашем MVC-приложении.

## 2.15.1 Совместная работа Представлений и Контроллеров

Как только какой-то определённый контроллер завершает свою работу, Phalcon автоматически передаёт управление компоненту представлений. Этот компонент ищет в папке представлений такое, название которое совпадает с последним исполнившимся контроллером, а затем — файл, имя которого соответствует последнему выполненному действию (action). Например, запрос по URL <http://127.0.0.1/blog/posts/show/301> Phalcon будет разбирать следующим образом:

|               |           |
|---------------|-----------|
| Адрес сервера | 127.0.0.1 |
| Папка Phalcon | blog      |
| Контроллер    | posts     |
| Действие      | show      |
| Параметр      | 301       |

Dispatcher будет искать “PostsController” и его метод “showAction”. Листинг простейшего контроллера для этого примера:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function showAction($postId)
    {
        // Передать параметр $postId в представление
        $this->view->setVar("postId", $postId);
    }
}
```

Метод setVar позволяет создавать переменные, которые могут быть использованы в шаблоне. В примере выше показано, как передать переменную \$postId.

## 2.15.2 Иерархия

*Phalcon|Mvc|View* поддерживает иерархическую файловую структуру. Это позволяет определять местонахождение как основных шаблонов (main layout), так и шаблонов для отдельных контроллеров (controller layout) с помощью задания имён для соответствующих им папок.

В качестве движка по умолчанию компонент использует сам PHP, поэтому представления должны иметь расширение .phtml. Если в качестве папки с представлениями используется *app/views*, то компонент автоматически будет искать следующие 3 файла.

| Название    | Файл                          | Описание   |
|-------------|-------------------------------|--|
| Action      | app/views/posts/show.phtml    | Представление, связанное с конкретным действием контроллера.                                   |
| View        |                               | Используется только при выполнении этого действия.   |
| Controller  | app/views/layouts/posts.phtml | Представление, связанное с контроллером. Используется для любого действия контроллера “posts”. |
| Layout      | app/views/index.phtml         | Основной шаблон приложения. Используется для любого контроллера или действия.                  |
| Main Layout |                               |  |

Совершенно не обязательно использовать все упомянутые выше файлы. *Phalcon|Mvc|View* просто

пропустит тот файл, которого нет, и перейдёт к следующему. Если же существуют все три файла представлений, то они будут обработаны следующим образом:

```
<!-- app/views/posts/show.phtml -->

<h3>This is show view!</h3>

<p>I have received the parameter <?php echo $postId ?></p>

<!-- app/views/layouts/posts.phtml -->

<h2>This is the "posts" controller layout!</h2>

<?php echo $this->getContent() ?>

<!-- app/views/index.phtml -->
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <h1>This is main layout!</h1>

        <?php echo $this->getContent() ?>

    </body>
</html>
```

Обратите внимание на строчки, в которых происходит вызов метода `$this->getContent()`. Он указывает на *Phalcon\Mvc\View*, где необходимо вывести результат, полученный при обработке представления, находящегося выше в иерархической структуре. Вывод для нашего примера будет представлять собой следующее:

align center

Сгенерированный HTML-код по этому запросу:

```
<!-- app/views/index.phtml -->
<html>
    <head>
        <title>Example</title>
    </head>
    <body>

        <h1>This is main layout!</h1>

        <!-- app/views/layouts/posts.phtml -->

        <h2>This is the "posts" controller layout!</h2>

        <!-- app/views/posts/show.phtml -->

        <h3>This is show view!</h3>

        <p>I have received the parameter 101</p>

    </body>
</html>
```



## Использование Шаблонов

Шаблоны — это представления, которые могут быть общими для разных действий контроллера. По сути они играют роль представлений контроллеров (controller layouts), поэтому их необходимо помещать папку layouts.

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{
    public function initialize()
    {
        $this->view->setTemplateAfter('common');
    }

    public function lastAction()
    {
        $this->flash->notice("These are the latest posts");
    }
}

<!-- app/views/index.phtml -->
<!DOCTYPE html>
<html>
    <head>
        <title>Blog's title</title>
    </head>
    <body>
```

```
<?php echo $this->getContent() ?>
</body>
</html>

<!-- app/views/layouts/common.phtml --&gt;

&lt;ul class="menu"&gt;
    &lt;li&gt;&lt;a href="/"&gt;Home&lt;/a&gt;&lt;/li&gt;
    &lt;li&gt;&lt;a href="/articles"&gt;Articles&lt;/a&gt;&lt;/li&gt;
    &lt;li&gt;&lt;a href="/contact"&gt;Contact us&lt;/a&gt;&lt;/li&gt;
&lt;/ul&gt;

&lt;div class="content"&gt;&lt;?php echo $this-&gt;getContent() ?&gt;&lt;/div&gt;

<!-- app/views/layouts/posts.phtml --&gt;

&lt;h1&gt;Blog Title&lt;/h1&gt;

&lt;?php echo $this-&gt;getContent() ?&gt;

<!-- app/views/posts/last.phtml --&gt;

&lt;article&gt;
    &lt;h2&gt;This is a title&lt;/h2&gt;
    &lt;p&gt;This is the post content&lt;/p&gt;
&lt;/article&gt;

&lt;article&gt;
    &lt;h2&gt;This is another title&lt;/h2&gt;
    &lt;p&gt;This is another post content&lt;/p&gt;
&lt;/article&gt;</pre>
```

Вывод получится следующим:

```
<!-- app/views/index.phtml -->
<!DOCTYPE html>
<html>
    <head>
        <title>Blog's title</title>
    </head>
    <body>

        <!-- app/views/layouts/common.phtml -->

        <ul class="menu">
            <li><a href="/">Home</a></li>
            <li><a href="/articles">Articles</a></li>
            <li><a href="/contact">Contact us</a></li>
        </ul>

        <div class="content">
            <!-- app/views/layouts/posts.phtml -->
            <h1>Blog Title</h1>
            <!-- app/views/posts/last.phtml -->
```

```

<article>
    <h2>This is a title</h2>
    <p>This is the post content</p>
</article>

<article>
    <h2>This is another title</h2>
    <p>This is another post content</p>
</article>

</div>

</body>
</html>

```

## Управление уровнями отрисовки (Rendering Levels)

Как уже говорилось выше, *Phalcon\ Mvc\ View* поддерживает иерархию представлений. Для управления уровнями отрисовки используется метод *PhalconMvc\ View::setRenderLevel()*.

Его можно вызвать в контроллере или вышестоящем уровне представления для изменения стандартного процесса отрисовки.

```

<?php

use Phalcon\Mvc\Controller,
    Phalcon\Mvc\View;

class PostsController extends Controller
{

    public function indexAction()
    {

    }

    public function findAction()
    {

        // Ajax-ответ, генерация представления не нужна
        $this->view->setRenderLevel(View::LEVEL_NO_RENDER);

        //...
    }

    public function showAction($postId)
    {
        // Показать только представление, относящееся к конкретному действию контроллера
        $this->view->setRenderLevel(View::LEVEL_ACTION_VIEW);
    }
}

```

Допустимые уровни отрисовки:

Константы	Описание	Порядок
LEVEL_NO_RENDER	Отключает генерацию каких-либо представлений.	1
LEVEL_ACTION_VIEW	Генерация представления, относящегося к конкретному действию.	
LEVEL_BEFORE_TEMPLATE	Генерация шаблонов представлений, предшествующих layout контроллера.	2
LEVEL_LAYOUT	Генерация представления, для layout контроллера.	3
LEVEL_AFTER_TEMPLATE	Генерация шаблонов представлений, следующих за layout контроллера.	4
LEVEL_MAIN_LAYOUT	Генерация представления для главного layout. Файл views/index.phtml	5

## Отключение уровней отрисовки

Если какие-то уровни не используются в приложении, их можно выключить для всего приложения:

```
<?php

use Phalcon\Mvc\View;

$di->set('view', function(){

    $view = new View();

    // Отключить несколько уровней
    $view->disableLevel(array(
        View::LEVEL_LAYOUT => true,
        View::LEVEL_MAIN_LAYOUT => true
    ));

    return $view;
}, true);
```

или только для какой-либо его части:

```
<?php

use Phalcon\Mvc\View,
    Phalcon\Mvc\Controller;

class PostsController extends Controller
{

    public function indexAction()
    {

    }

    public function findAction()
    {
        $this->view->disableLevel(View::LEVEL_MAIN_LAYOUT);
    }
}
```

### 2.15.3 Переопределение Представлений (Picking Views)

Как уже упоминалось выше, *Phalcon\Mvc\View*, работающий под управлением *Phalcon\Mvc\Application*, по умолчанию будет использовать представления соответствующие последним выполнившимся контроллеру и действию. Это можно переопределить с помощью метода *Phalcon\Mvc\View::pick()*:

```
<?php

class ProductsController extends \Phalcon\Mvc\Controller
{

    public function listAction()
    {
        // Использовать для отрисовки "views-dir/products/search"
        $this->view->pick("products/search");

        // Использовать для отрисовки "views-dir/products/list"
        $this->view->pick(array('products'));

        // Использовать для отрисовки "views-dir/products/search"
        $this->view->pick(array(1 => 'search'));
    }

}
```

### 2.15.4 Отключение представления

Если в контроллере нет никакого вывода, то отключить компонент представления, чтобы избежать выполнение ненужных действий:

```
<?php

class UsersController extends \Phalcon\Mvc\Controller
{

    public function closeSessionAction()
    {
        // Тут завершилась сессия
        //...

        // HTTP редирект
        $this->response->redirect('index/index');

        // Отключение компонента представлений
        $this->view->disable();
    }

}
```

Вы можете вернуть объект ‘*response*’, чтобы вручную отключить компонент представления:

```
<?php

class UsersController extends \Phalcon\Mvc\Controller
{
```

```
public function closeSessionAction()
{
    //Close session
    //...

    //An HTTP Redirect
    return $this->response->redirect('index/index');
}

}
```

## 2.15.5 Простая отрисовка

*Phalcon\Mvc\View\Simple* — это аналогичный *Phalcon\Mvc\View* компонент. Он сохраняет основной подход *Phalcon\Mvc\View*, но не реализует иерархию файлов, что, по сути, является основной особенностью его коллеги.

Этот компонент позволяет разработчику определять какой файл представления использовать и где он находится. Кроме того, компонент может использовать структуру наследования в шаблонизаторе *Volt* и ему подобных.

Компонент представлений по-умолчанию может быть замещён в контейнере сервисов:

```
<?php

$di->set('view', function() {

    $view = new Phalcon\Mvc\View\Simple();

    $view->setViewsDir('../app/views/');

    return $view;

}, true);
```

Процесс автоматической отрисовки может быть отключен в *Phalcon\Mvc\Application* (если это необходимо):

```
<?php

try {

    $application = new Phalcon\Mvc\Application($di);

    $application->useImplicitView(false);

    echo $application->handle()->getContent();

} catch (\Exception $e) {
    echo $e->getMessage();
}
```

Для отрисовки необходимо вызвать метод `render`, указав конкретный путь к представлению, которое необходимо отрисовать:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
```

```
{
    public function indexAction()
    {
        //Render 'views-dir/index.phtml'
        echo $this->view->render('index');

        //Render 'views-dir/posts/show.phtml'
        echo $this->view->render('posts/show');

        //Render 'views-dir/index.phtml' passing variables
        echo $this->view->render('index', array('posts' => Posts::find()));

        //Render 'views-dir/posts/show.phtml' passing variables
        echo $this->view->render('posts/show', array('posts' => Posts::find()));
    }
}
```

## 2.15.6 Части шаблонов (Partials)

Части шаблонов (Partial templates) — это ещё один способ дробления процесса отрисовки на более простые части, которые впоследствии могут быть использованы в различных частях приложения. С помощью них можно вынести код отрисовки какой-то конкретной части шаблона в отдельный файл.

Использование части шаблонов аналогично использованию подпрограмм: детали реализации выносятся из представления с целью сделать код более простым и понятным. Например, вы могли бы получить такой шаблон:

```
<div class="top"><?php $this->partial("shared/ad_banner") ?></div>

<div class="content">
    <h1>Robots</h1>

    <p>Check out our specials for robots:</p>
    ...
</div>

<div class="footer"><?php $this->partial("shared/footer") ?></div>
```

Метод partial() принимает в качестве второго параметра массив переменных, которые будут доступны только в пределах части шаблона:

```
<?php $this->partial("shared/ad_banner", array('id' => $site->id, 'size' => 'big')) ?>
```

## 2.15.7 Передача переменных контроллера

*Phalcon | Mvc | View* позволяет использовать в каждом контроллере переменную компонента представления (`$this->view`). Её можно использовать, чтобы устанавливать значения переменных представления непосредственно в действиях контроллера. Для этого используется метод `setVar()`.

```
<?php
```

```
<?php
```

```
class PostsController extends \Phalcon\Mvc\Controller
{
    public function indexAction()
    {
    }

    public function showAction()
    {
        // Передаёт все посты во представление
        $this->view->setVar("posts", Posts::find());

        // Используется "магический" септер
        $this->view->posts = Posts::find();

        // Передача сразу нескольких переменных с помощью массива
        $this->view->setVars(array(
            'title' => $post->title,
            'content' => $post->content
        ));
    }
}
```

Первым параметром метода setVar() передаётся название переменной, которая будет создана и может быть использована в представлении. Эта переменная может быть любого типа, как простым, например, строкой или числом, так и сложной структурой вроде массива или коллекции объектов.

```
<div class="post">
<?php

foreach ($posts as $post) {
    echo "<h1>", $post->title, "</h1>";
}

?>
</div>
```

## 2.15.8 Использование моделей в слое представления

Модели приложения всегда доступны из слоя представления. Во время исполнения *Phalcon\Loader* автоматически создаёт их копии:

```
<div class="categories">
<?php

foreach (Categories::find("status = 1") as $category) {
    echo "<span class='category'>", $category->name, "</span>";
}

?>
</div>
```

Хотя и существует возможность вызова таких методов модели, как insert() или update(), это не рекомендуется, так как при этом становится невозможной передача выполнения другому контроллеру в случае возникновения ошибки или исключительной ситуации.

## 2.15.9 Кэширование фрагментов Представления

При разработке динамических веб-сайтов некоторые их области могут изменяться достаточно редко и результат вывода будет совпадать для похожих запросов. Для увеличения производительности *Phalcon|Mvc|View* предоставляет возможность кэширования частей или даже всего результата отрисовки.

*Phalcon|Mvc|View* используется совместно с *Phalcon|Cache*, для обеспечения простой способ кэширования частей вывода. Вы можете вручную установить обработчик кэша или глобальный обработчик:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function showAction()
    {
        // Кэширование с использованием настроек по умолчанию
        $this->view->cache(true);
    }

    public function showArticleAction()
    {
        // Кэширование на один час
        $this->view->cache(array(
            "lifetime" => 3600
        ));
    }

    public function resumeAction()
    {
        // Кэширование представления этого действия на один день с ключем "resume-cache"
        $this->view->cache(
            array(
                "lifetime" => 86400,
                "key"      => "resume-cache",
            )
        );
    }

    public function downloadAction()
    {
        // Использование стороннего сервиса для кэширования
        $this->view->cache(
            array(
                "service"  => "myCache",
                "lifetime" => 86400,
                "key"      => "resume-cache",
            )
        );
    }
}
```

Если ключ кэша не задан, то компонент автоматически создаёт его на основе `md5` суммы имени текущего представления. Это хороший способ задания уникального ключа для кэша конкретного представления.

Когда компонент Представления должен что-то закэшировать, он запрашивает сервис кэша у контейнера

нера сервисов. По соглашению этот сервис называется “viewCache”:

```
<?php

use Phalcon\Cache\Frontend\Output as OutputFrontend,
    Phalcon\Cache\Backend\Memcache as MemcacheBackend;

// Назначение сервиса кэширования представлений
$di->set('viewCache', function() {

    // Кэширование данных на сутки по умолчанию
    $frontCache = new OutputFrontend(array(
        "lifetime" => 86400
    ));

    // Настройки соединения с Memcached
    $cache = new MemcacheBackend($frontCache, array(
        "host" => "localhost",
        "port" => "11211"
    ));

    return $cache;
});
```

Интерфейс всегда должен быть Phalcon\Cache\Frontend\Output, а сервис “viewCache” должен быть зарегистрирован как всегда открытый (not shared) в контейнере сервисов (DI).

Использование кэширования представлений бывает полезно, чтобы избежать выполнение действий контроллеров, направленных на получение данных, которые используются для отображения в представлениях.

Для достижения этой цели необходимо однозначно идентифицировать каждый кэш с помощью ключа. Прежде чем выполнять вычисления или запросы для отображаемых в представлении данных, необходимо убедиться, что кэш не существует или его срок истек:

```
<?php

class DownloadController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

        // Проверяет, кэш с ключом "downloads" на существование или истёкший срок
        if ($this->view->getCache()->exists('downloads')) {

            // Запрос последних загрузок
            $latest = Downloads::find(array(
                'order' => 'created_at DESC'
            ));

            $this->view->latest = $latest;
        }

        // Включает кэширование с ключом "downloads"
        $this->view->cache(array(
            'key' => 'downloads'
        ));
    }
}
```

```
}
```

Пример реализации кэширования фрагментов — PHP alternative site.

## 2.15.10 Шаблонизаторы

Шаблонизаторы помогают дизайнерам создавать представления без использования сложного синтаксиса. Phalcon имеет встроенный мощный и одновременно быстрый шаблонизатор *Volt*.

Кроме того, *Phalcon|Mvc|View* позволяет использовать другие шаблонизаторы вместо обычного PHP или Volt.

Использование различных шаблонизаторов, как правило, требует сложного разбора кода с применением внешних PHP-библиотек, генерирующих результат для пользователя. Это, в свою очередь, увеличивает количество ресурсов, используемых приложением.

Если используется внешний шаблонизатор, *Phalcon|Mvc|View* обеспечивает иерархию файловой структуры и по-прежнему предоставляет доступ к API из этих шаблонов, но с чуть большими затратами.

Этот компонент использует адаптеры, что позволяет Phalcon общаться с внешними шаблонизаторами единым образом. Рассмотрим, как это происходит.

### Создание собственного адаптера для шаблонизатора

Существует множество шаблонизаторов, которые вы можете подключить или создать свой собственный. Первый шаг к использованию внешнего шаблонизатора — это создание адаптера для него.

Адаптер шаблонизатора — это класс, который служит мостом *Phalcon|Mvc|View* и самим шаблонизатором. Обычно необходимо реализовать всего два метода: `_construct()` и `render()`. В первый передаются экземпляры *Phalcon|Mvc|View* и контейнер DI, используемый в приложении.

Во второй — абсолютный путь к файлу представления и параметры, устанавливаемые с помощью `$this->view->setVar()`. Их можно использовать, как только в них появится необходимость.

```
<?php
```

```
class MyTemplateAdapter extends \Phalcon\Mvc\View\Engine
{

    /**
     * Конструктор адаптера
     *
     * @param \Phalcon\Mvc\View $view
     * @param \Phalcon\DI $di
     */
    public function __construct($view, $di)
    {
        // Инициализация адаптера
        parent::__construct($view, $di);
    }

    /**
     * Отрисовывает представление с помощью шаблонизатора
     *
     * @param string $path
     * @param array $params
     */
}
```

```
public function render($path, $params)
{
    // Доступ к view
    $view = $this->_view;

    // Доступ к настройкам
    $options = $this->_options;

    // Render the view
    //...
}

}
```

## Изменение шаблонизатора

Вы можете изменить или дополнить шаблонизатор из контроллера следующим образом:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {
        // Назначение шаблонизатора
        $this->view->registerEngines(
            array(
                ".my-html" => "MyTemplateAdapter"
            )
        );
    }

    public function showAction()
    {
        // Использование нескольких шаблонизаторов
        $this->view->registerEngines(
            array(
                ".my-html" => 'MyTemplateAdapter',
                ".phtml" => '\Phalcon\Mvc\View\Engine\Php',
            )
        );
    }
}
```

Вы можете полностью заменить шаблонизатор или использовать несколько шаблонизаторов одновременно. Метод `\Phalcon\Mvc\View::registerEngines()` принимает в качестве параметра массив, в котором описываются данные шаблонизаторов. Ключами массива в этом случае будут расширения файлов, что помогает отличить их друг от друга. Файлы шаблонов, относящиеся к этим шаблонизаторам должны иметь соответствующие расширения.

Порядок выполнения шаблонизаторов определяется порядком, в котором они описаны в `\Phalcon\Mvc\View::registerEngines()`. Если `\Phalcon\Mvc\View` обнаружит два представления с одинаковым именами, но разными расширениями, то он отрисует тот, который был указан первым.

Если вы хотите зарегистрировать шаблонизатор или назначить его для любого запроса в приложении, вы можете сделать это при создании сервиса представления:

```
<?php

// Настройка компонента представления
$di->set('view', function() {

    $view = new \Phalcon\Mvc\View();

    // A trailing directory separator is required
    $view->setViewsDir('../app/views/');

    $view->registerEngines(array(
        ".my-html" => 'MyTemplateAdapter',
    ));

    return $view;
}, true);
```

Адаптеры для некоторых шаблонизаторов можно найти здесь: [Phalcon Incubator](#).

### 2.15.11 Внедрение сервисов в Представление

Каждое представление, исполняемое внутри экземпляра *Phalcon|DI|Injectable* получает простой доступ к сервисам приложения.

Следующий пример демонстрирует как можно написать ajax request на jQuery используя url из фреймворка. Сервис “url” (обычно это *Phalcon|Mvc|Url*) внедрён в представление и доступен как свойство с таким же именем:

```
<script type="text/javascript">

$.ajax({
    url: "<?php echo $this->url->get("cities/get") ?>"
})
.done(function() {
    alert("Done!");
});

</script>
```

### 2.15.12 Отдельное использование компонента

Все компоненты в Phalcon могут быть использованы по-отдельности благодаря их слабой связи друг с другом. Ниже приводится пример самостоятельного использования *Phalcon|Mvc|View*:

#### Иерархическая отрисовка

Использование *Phalcon|Mvc|View* в качестве самостоятельного компонента:

```
<?php

$view = new \Phalcon\Mvc\View();
```

```
// A trailing directory separator is required
$view->setViewsDir("../app/views/");

// Передача переменных в представление
$view->setVar("someProducts", $products);
$view->setVar("someFeatureEnabled", true);

// Начало буферизации вывода
$view->start();

// Отрисовка всей иерархии представлений, связанной с products/list.phtml
$view->render("products", "list");

// Конец буферизации вывода
$view->finish();

echo $view->getContent();
```

Так же доступен короткий синтаксис:

```
<?php

$view = new \Phalcon\Mvc\View();

echo $view->getRender('products', 'list',
    array(
        "someProducts" => $products,
        "someFeatureEnabled" => true
    ),
    function($view) {
        // Установка дополнительных опций
        $view->setViewsDir("../app/views/");
        $view->setRenderLevel(\Phalcon\Mvc\View::LEVEL_LAYOUT)
    }
);
```

## Простая отрисовка

Использование *Phalcon|Mvc|View|Simple* в качестве самостоятельного компонента:

```
<?php

$view = new \Phalcon\Mvc\View\Simple();

// Обязательно закрывающий слеш
$view->setViewsDir("../app/views/");

// Возвращает результат отрисовки в виде строки
echo $view->render("templates/welcomeMail");

// Передача параметров для отрисовки
echo $view->render("templates/welcomeMail", array(
    'email' => $email,
    'content' => $content
));
```

### 2.15.13 События компонента представлений

*Phalcon\Mvc\View* и *Phalcon\Mvc\View* могут отправлять события *EventsManager*, если последний представлен. Тип событий — “view”. Некоторые из них, возвращая булевое значение `false` могут остановить текущую операцию. Поддерживаются следующие события:

Названия события	Условия срабатывания	Могут ли остановить операцию?
<code>beforeRender</code>	Перед началом процесса отрисовки	Да
<code>beforeRenderView</code>	Перед отрисовкой существующего представления	Да
<code>afterRenderView</code>	После отрисовки существующего представления	Нет
<code>afterRender</code>	После завершения процесса отрисовки	Нет
<code>notFoundView</code>	Если представление не найдено	Нет

Пример ниже демонстрирует как назначить слушателей (listeners) для этого компонента:

```
<?php

$di->set('view', function() {

    // Создание обработчика событий
    $eventsManager = new Phalcon\Events\Manager();

    // Назначение слушателя для событий типа "view"
    $eventsManager->attach("view", function($event, $view) {
        echo $event->getType(), ' - ', $view->getActiveRenderPath(), PHP_EOL;
    });

    $view = new \Phalcon\Mvc\View();
    $view->setViewsDir("../app/views/");

    // Назначение обработчика событий для компонента представления
    $view->setEventsManager($eventsManager);

    return $view;
}, true);
```

Следующий пример показывает, как создать плагин, который очищает/исправляет HTML, сгенерированный с использованием *Tidy*:

```
<?php

class TidyPlugin
{

    public function afterRender($event, $view)
    {

        $tidyConfig = array(
            'clean' => true,
            'output-xhtml' => true,
            'show-body-only' => true,
            'wrap' => 0,
        );

        $tidy = tidy_parse_string($view->getContent(), $tidyConfig, 'UTF8');
    }
}
```

```
$tidy->cleanRepair();

$view->setContent((string) $tidy);
}

// Назначение плагина в качестве слушателя
$eventsManager->attach("view:afterRender", new TidyPlugin());
```

## 2.16 Помощники представлений

Написание и хранение HTML разметки может быстро надоест из-за многочисленных элементов с различными именами и атрибутами, зависимостями которые должны быть всегда учтены. Phalcon облегчает эти сложности предлагая для работы компонент [Phalcon\Tag](#), предоставляющий помощников (helpers) для работы с элементами и формирования готовой HTML разметки.

Компонент может быть использован в шаблонах представлений из HTML+PHP или в шаблонизаторе [Volt](#).

Это руководство не отображает все возможности помощников представлений и их аргументов. Для получения полной актуальной информации ознакомьтесь с документацией по API [Phalcon\Tag](#).

### 2.16.1 Типы документов

Phalcon содержит помощника `Phalcon\Tag::setDoctype()` для установки типа документа. Установка типа документа влияет на формирование компонентом HTML разметки. Например, при указании любого типа из вариантов XHTML Phalcon будет добавлять в HTML код элементов закрывающие тэги, именно так как того требует стандарт XHTML.

Разрешёнными константами для `Phalcon\Tag` являются:

Константа	Тип документа
<code>HTML32</code>	HTML 3.2
<code>HTML401_STRICT</code>	HTML 4.01 Strict
<code>HTML401_TRANSITIONAL</code>	HTML 4.01 Transitional
<code>HTML401_FRAMESET</code>	HTML 4.01 Frameset
<code>HTML5</code>	HTML 5
<code>XHTML10_STRICT</code>	XHTML 1.0 Strict
<code>XHTML10_TRANSITIONAL</code>	XHTML 1.0 Transitional
<code>XHTML10_FRAMESET</code>	XHTML 1.0 Frameset
<code>XHTML11</code>	XHTML 1.1
<code>XHTML20</code>	XHTML 2.0
<code>XHTML5</code>	XHTML 5

Установка типа документа:

```
<?php $this->tag->setDoctype(\$this->tag->HTML401_STRICT); ?>
```

Вывод типа документа:

```
<?= $this->tag->getDoctype() ?>
<html>
```

```
<!-- your HTML code -->
</html>
```

Из кода выше сформируется такой HTML результат:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<!-- your HTML code -->
</html>
```

Синтаксис Volt:

```
{{ get_doctype() }}
<html>
<!-- your HTML code -->
</html>
```

## 2.16.2 Создание ссылок

Наиболее частая задача в работе любого веб-приложения - формирование ссылок для перехода на другие страницы. Если ссылки внутренние (для этого же приложения), то их можно формировать следующим образом:

```
<!-- для стандартных путей -->
<?= $this->tag->linkTo("products/search", "Search") ?>

<!-- используя CSS -->
<?= $this->tag->linkTo(array('products/edit/10', 'Edit', 'class' => 'edit-btn')) ?>

<!-- для именованных маршрутов -->
<?= $this->tag->linkTo(array('for' => 'show-product', 'title' => 123, 'name' => 'carrots'), 'Show') ?>
```

На самом деле, все такие ссылки формируются с использованием компонент *Phalcon\ Mvc\Url* (или другого используемого для “url” сервиса )

Аналогично ссылки формируются в Volt:

```
<!-- for the default route -->
{{ link_to("products/search", "Search") }}

<!-- for a named route -->
{{ link_to(['for': 'show-product', 'id': 123, 'name': 'carrots'], 'Show') }}
```

## 2.16.3 Создание форм

Формы веб-приложений играют важную роль для получения данных, введённых пользователем. Пример ниже показывает вариант реализации формы поиска с использованием помощников представлений.

```
<!-- Отправка формы методом POST -->
<?= $this->tag->form("products/search") ?>
    <label for="q">Search:</label>
    <?= $this->tag->textField("q") ?>
    <?= $this->tag->submitButton("Search") ?>
</form>
```

```
<!-- Использование специфичного для элемента FORM тега - метода отправки данных -->
<?= $this->tag->form(array("products/search", "method" => "get")); ?>
<label for="q">Search:</label>
<?= $this->tag->textField("q"); ?>
<?= $this->tag->submitButton("Search"); ?>
</form>
```

Из кода выше сформируется такой HTML результат:

```
<form action="/store/products/search/" method="get">
    <label for="q">Search:</label>
    <input type="text" id="q" value="" name="q" />
    <input type="submit" value="Search" />
</endform>
```

Аналогичную форму можно сгенерировать в Volt:

```
<!-- Specifying another method or attributes for the FORM tag -->
{{ form("products/search", "method": "get") }}
    <label for="q">Search:</label>
    {{ text_field("q") }}
    {{ submit_button("Search") }}
</form>
```

Phalcon так же содержит *сборщик форм* для создания форм с использованием объектно-ориентированного подхода.

## 2.16.4 Помощники создания элементов форм

Phalcon предоставляет ряд помощников для создания элементов формы, такие как текстовые поля, кнопки и многие другие. Первый параметр в таких методах это всегда имя элемента. При отправке формы это имя будет передаваться вместе со значениями формы. В контроллере вы можете получить значение элемента используя это же имя элемента и методы `getPost()` и `getQuery()` объекта запроса (`$this->request`).

```
<?php echo $this->tag->textField("username") ?>

<?php echo $this->tag->textArea(array(
    "comment",
    "This is the content of the text-area",
    "cols" => "6",
    "rows" => 20
)) ?>

<?php echo $this->tag->passwordField(array(
    "password",
    "size" => 30
)) ?>

<?php echo $this->tag->hiddenField(array(
    "parent_id",
    "value"=> "5"
)) ?>
```

Синтаксис Volt:

```

{{ text_field("username") }}

{{ text_area("comment", "This is the content", "cols": "6", "rows": 20) }}

{{ password_field("password", "size": 30) }}

{{ hidden_field("parent_id", "value": "5") }}

```

## 2.16.5 Создание выпадающих списков

Работать с выпадающими списками легко при хранении данных для их формирования в виде ассоциативных массивов PHP. У Phalcon имеется два помощника для работы с такими списками - Phalcon\Tag::select() и Phalcon\Tag::selectStatic(). Метод Phalcon\Tag::select() был специально разработан для работы с *Phalcon\ Mvc\ Model*, а Phalcon\Tag::selectStatic() с PHP массивами.

```

<?php

// Используем данные из resultset
echo $this->tag->select(
    array(
        "productId",
        Products::find("type = 'vegetables'"),
        "using" => array("id", "name")
    )
);

// Используем данные из массива
echo $this->tag->selectStatic(
    array(
        "status",
        array(
            "A" => "Active",
            "I" => "Inactive",
        )
    )
);

```

Сформируется такой HTML:

```

<select id="productId" name="productId">
    <option value="101">Tomato</option>
    <option value="102">Lettuce</option>
    <option value="103">Beans</option>
</select>

<select id="status" name="status">
    <option value="A">Active</option>
    <option value="I">Inactive</option>
</select>

```

Так же можно добавить пустой - “empty” блок в HTML:

```

<?php

// Формирование выпадающего списка с пустым элементом
echo $this->tag->select(
    array(

```

```
        "productId",
        Products::find("type = 'vegetables'"),
        "using" => array("id", "name"),
        "useEmpty" => true
    )
);
```

Получится HTML

```
<select id="productId" name="productId">
    <option value="">Choose..</option>
    <option value="101">Tomato</option>
    <option value="102">Lettuce</option>
    <option value="103">Beans</option>
</select>

<?php

// Указание параметров пустого элемента
echo $this->tag->select(
    array(
        'productId',
        Products::find("type = 'vegetables'"),
        'using' => array('id', 'name'),
        'useEmpty' => true,
        'emptyText' => 'Выберите значение...',
        'emptyValue' => '@'
    ),
);

<select id="productId" name="productId">
    <option value="@>Выберите значение...</option>
    <option value="101">Tomato</option>
    <option value="102">Lettuce</option>
    <option value="103">Beans</option>
</select>
```

Аналогичный пример для Volt:

```
{# Creating a Select Tag with an empty option with default text #}
{{ select('productId', products, 'using': ['id', 'name'],
    'useEmpty': true, 'emptyText': 'Please, choose one...', 'emptyValue': '@') }}
```

## 2.16.6 Установка HTML атрибутов

Все помощники (helpers) могут принимать в качестве первого параметра массив, в котором можно указывать атрибуты для формирования HTML кода элемента:

```
<?php \$this->tag->textField(
    array(
        "price",
        "size"      => 20,
        "maxlength" => 30,
        "placeholder" => "Введите цену",
    )
) ?>
```

так же и в Volt:

```
{{ text_field("price", "size": 20, "maxlength": 30, "placeholder": "Enter a price") }}
```

Сформированный HTML:

```
<input type="text" name="price" id="price" size="20" maxlength="30"
placeholder="Enter a price" />
```

## 2.16.7 Установка значений помощников

### Из контроллера

Установка значений форм в контроллерах является хорошей практикой в парадигме MVC. Вы можете устанавливать значения в контроллерах используя метод `Phalcon\Tag::setDefault()`. Этот помощник устанавливает значения по умолчанию для элементов форм, используемых в представлениях. При выводе формы производится проверка на предустановленные значения, и если значение не указано напрямую в помощнике, то будет использовано указанное в контроллере.

```
<?php

class ProductsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {
        $this->tag->setDefault("color", "Blue");
    }

}
```

В представлении помощник `selectStatic` сделает активным установленный индекс. В данном случае это “цвет”:

```
<?php

echo \$this->tag->selectStatic(
    array(
        "color",
        array(
            "Yellow" => "Yellow",
            "Blue"   => "Blue",
            "Red"    => "Red"
        )
    )
);
```

В результате будет сформирован HTML код выпадающего списка с выбранным значением “Blue”:

```
<select id="color" name="color">
    <option value="Yellow">Yellow</option>
    <option value="Blue" selected="selected">Blue</option>
    <option value="Red">Red</option>
</select>
```

## Из запроса (Request)

Одна из волшебных функций Phalcon реализованной в компоненте [Phalcon\Tag](#) позволяет хранить данные, введённые в формы, между запросами. Таким образом, вы можете легко выводить сообщения об ошибках и правильности заполнения формы без потери введенных пользователем данных.

### Установка значений напрямую

Каждый помощник форм поддерживает параметр “value”, с помощью него указываются конечные значения элемента. При указании этого параметра все остальные предустановленные методом `setDefault()` значения будут проигнорированы.

#### 2.16.8 Изменение title документа

[Phalcon\Tag](#) содержит так же помощника для динамического изменения названия (`title`) документа в контроллерах. Использование такого варианта показано в примере.

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function initialize()
    {
        $this->tag->setTitle(" Суперсайт");
    }

    public function indexAction()
    {
        $this->tag->prependTitle("Главная страница - ");
    }

}

<html>
    <head>
        <?php echo \$this->tag->getTitle(); ?>
    </head>
    <body>

    </body>
</html>
```

Результат:

```
<html>
    <head>
        <title>Главная страница - Суперсайт</title>
    </head>
    <body>

    </body>
</html>
```

## 2.16.9 Помощники работы со статичными элементами

*Phalcon\Tag* так же содержит помощников для генерации тегов script, link и img. Они помогают в быстрой и простой генерации тегов подключения статичных ресурсов.

### Изображения

```
<?php

// Сформируется 
echo \$this->tag->image("img/hello.gif");

// Сформируется 
echo \$this->tag->image(
    array(
        "img/hello.gif",
        "alt" => "alternative text"
    )
);
```

Использование в Volt:

```
{# Сформируется  #}
{{ image("img/hello.gif") }}

{# Сформируется  #}
{{ image("img/hello.gif", "alt": "alternative text") }}
```

### Таблицы стилей (Stylesheets)

```
<?php

// Сформируется <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Rosario" type="text/css">
echo \$this->tag->stylesheetLink("http://fonts.googleapis.com/css?family=Rosario", false);

// Сформируется <link rel="stylesheet" href="/your-app/css/styles.css" type="text/css">
echo \$this->tag->stylesheetLink("css/styles.css");
```

Аналогично в Volt:

```
{# Сформируется <link rel="stylesheet" href="http://fonts.googleapis.com/css?family=Rosario" type="text/css"> #}
{{ stylesheet_link("http://fonts.googleapis.com/css?family=Rosario", false) }}

{# Сформируется <link rel="stylesheet" href="/your-app/css/styles.css" type="text/css"> #}
{{ stylesheet_link("css/styles.css") }}
```

### Javascript

```
<?php

// Сформируется <script src="http://localhost/javascript/jquery.min.js" type="text/javascript"></script>
echo \$this->tag->javascriptInclude("http://localhost/javascript/jquery.min.js", false);

// Сформируется <script src="/your-app/javascript/jquery.min.js" type="text/javascript"></script>
echo \$this->tag->javascriptInclude("javascript/jquery.min.js");
```

То же самое в Volt:

```
{# Сформируется <script src="http://localhost/javascript/jquery.min.js" type="text/javascript"></script> #}
{{ javascript_include("http://localhost/javascript/jquery.min.js", false) }}

{# Сформируется <script src="/your-app/javascript/jquery.min.js" type="text/javascript"></script> #}
{{ javascript_include("javascript/jquery.min.js") }}
```

## Элементы HTML5 - общий HTML помощник

Фалкон содержит HTML помощник, позволяющий генерировать любой HTML элемент. Он полностью зависит от разработчика, необходимо лишь название элемента.

```
<?php

// Generate
// <canvas id="canvas1" width="300" class="cnvclass">
// This is my canvas
// </canvas>
echo \$this->tag->tagHtml("canvas", array("id" => "canvas1", "width" => "300", "class" => "cnvclass", false, true,
echo "This is my canvas";
echo \$this->tag->tagHtmlClose("canvas");
```

Синтаксис Volt:

```
{# Generate #}
{# <canvas id="canvas1" width="300" class="cnvclass"> #}
{# This is my canvas #}
{# </canvas> #}
{{ tag_html("canvas", ["id":"canvas1", "width":"300", "class":"cnvclass"], false, true, true) }}
This is my canvas
{{ tag_html_close("canvas") }}
```

## 2.16.10 Сервис Tag

*Phalcon|Tag* доступен через сервис ‘tag’, это значит, что вы можете получить доступ из любой части приложения, где есть доступ к контейнеру:

```
<?php echo $this->tag->linkTo('pages/about', 'About') ?>
```

Вы можете легко добавить свои хелперы в пользовательском компоненте, заменяющем сервис ‘tag’ в контейнере зависимостей:

```
<?php

class MyTags extends \Phalcon\Tag
{
    //...

    // Создаем новый хелпер
    static public function myAmazingHelper($parameters)
    {
        //...
    }

    // Переопределяем уже существующий метод
}
```

```

static public function textField($parameters)
{
    //...
}
}

```

Затем меняем сервис “tag”:

```

<?php

$di['tag'] = function() {
    return new MyTags();
};

```

### 2.16.11 Создание собственных помощников

Вы можете с лёгкостью создавать своих помощников расширяя *Phalcon\Tag* и реализуя собственных помощников. Пример ниже отображает вариант такой реализации:

```

<?php

class MyTags extends \Phalcon\Tag
{

    /**
     * Создёт виджет вывода тега HTML5 audio
     *
     * @param array
     * @return string
     */
    static public function audioField($parameters)
    {

        // Приведение к массиву
        if (!is_array($parameters)) {
            $parameters = array($parameters);
        }

        // Определение атрибутов "id" и "name"
        if (!isset($parameters[0])) {
            $parameters[0] = $parameters["id"];
        }

        $id = $parameters[0];
        if (!isset($parameters["name"])) {
            $parameters["name"] = $id;
        } else {
            if (!$parameters["name"]) {
                $parameters["name"] = $id;
            }
        }

        // Определение значения элемента
        // !$this->tag->setDefault() позволяет установить значение элемента
        if (isset($parameters["value"])) {
            $value = $parameters["value"];
            unset($parameters["value"]);
        }
    }
}

```

```
    } else {
        $value = self::getValue($id);
    }

    // Генерация кода
    $code = '<audio id="'. $id .'" value="'. $value .'';
    foreach ($parameters as $key => $attributeValue) {
        if (!is_integer($key)) {
            $code .= $key . '=' . $attributeValue . ' ';
        }
    }
    $code .= "/>";

    return $code;
}
```

Так же предлагаем вам ознакомиться с [Volt](#) - очень быстрым шаблонизатором для PHP. В нём вы же можете использовать возможности Phalcon\Tag в более дружественном синтаксисе.

## 2.17 Управление ресурсами (Assets Management)

Phalcon\Assets - это компонент позволяющий разработчику управлять статичными ресурсами в веб-приложении, такими как каскадные таблицы стилей или javascript'ы.

*Phalcon Assets Manager* доступен в контейнере сервисов, т.ч. вы можете добавлять ресурсы из любой части приложения, где он доступен.

### 2.17.1 Добавление ресурсов

Поддерживаются ресурсы двух типов: каскадные таблицы стилей и javascript'ы, но при необходимости, можете создать и другие. Внутренний механизм менеджера ресурсов хранит две коллекции, одну для javascript'ов, а другую для каскадных таблиц стилей.

Добавить ресурсы в эти коллекции очень просто:

```
<?php

class IndexController extends Phalcon\Mvc\Controller
{
    public function index()
    {
        // Добавляем некоторые локальные таблицы стилей
        $this->assets
            ->addCss('css/style.css')
            ->addCss('css/index.css');

        // и javascript'ы
        $this->assets
            ->addJs('js/jquery.js')
            ->addJs('js/bootstrap.min.js');
    }
}
```

```

    }
}

```

Далее, добавленные ресурсы могут быть отображены в представлениях:

```

<html>
    <head>
        <title>Некоторый удивительный веб-сайт</title>
        <?php $this->assets->outputCss() ?>
    </head>
    <body>

        <!-- ... -->

        <?php $this->assets->outputJs() ?>
    </body>
<html>

```

## 2.17.2 Локальные/удаленные ресурсы

Локальные ресурсы это те, которые предоставляются вами в том же приложении. Ссылки на локальные ресурсы генерируются с помощью сервиса ‘url’, чаще с применением [Phalcon\Mvc\Url](#).

Удаленные ресурсы, такие как общие библиотеки, на подобии jquery, bootstrap или пр. предоставляемые посредством CDN.

```

<?php

// Добавляем некоторые локальные и удаленные ресурсы
$this->assets
    ->addCss('//netdna.bootstrapcdn.com/twitter-bootstrap/2.3.1/css/bootstrap-combined.min.css', false)
    ->addCss('css/style.css', true);

```

## 2.17.3 Коллекции

В коллекциях группируются однотипные ресурсы. Менеджер ресурсов безоговорочно создает две: css и js. Для группирования специфичных ресурсов вы можете создавать дополнительные:

```

<?php

// Javascript'ы в заголовке
$this->assets
    ->collection('header')
    ->addJs('js/jquery.js')
    ->addJs('js/bootstrap.min.js');

// Javascript'ы в "подвале"
$this->assets
    ->collection('footer')
    ->addJs('js/jquery.js')
    ->addJs('js/bootstrap.min.js');

```

затем в представлении:

```

<html>
    <head>

```

```
<title>Некоторый удивительный веб-сайт</title>
<?php $this->assets->outputJs('header') ?>
</head>
<body>

<!-- ... -->

<?php $this->assets->outputJs('footer') ?>
</body>
<html>
```

## 2.17.4 Префиксы

К коллекциям могут применяться URL префиксы, это позволит в любой момент легко изменить расположение ресурсов с одного сервера на другой:

```
<?php

$scripts = $this->assets->collection('footer');

if ($config->enviroment == 'development') {
    $scripts->setPrefix('/');
} else {
    $scripts->setPrefix('http://cdn.example.com/');
}

$scripts->addJs('js/jquery.js')
    ->addJs('js/bootstrap.min.js');
```

Также, доступен синтаксис цепочки (chainable):

```
<?php

$scripts = $assets
    ->collection('header')
    ->setPrefix('http://cdn.example.com/')
    ->setLocal(false)
    ->addJs('js/jquery.js')
    ->addJs('js/bootstrap.min.js');
```

## 2.17.5 Минимизация/ Фильтрация

PhalconAssets предоставляет встроенную возможность минимизации javascript и CSS. Разработчик может создать коллекцию ресурсов с указаниями для Assets Manager, к каким ресурсам должны быть применены фильтры, а к каким нет. В дополнении к вышесказанному, “Jsmin” Дугласа Крокфорда (Douglas Crockford) входит в состав ядра минимизации javascript для увеличения производительности. Для минимизации CSS используется “CSSMin” Райна Дэя (Ryan Day).

Следующий пример показывает, как минимизировать набор ресурсов:

```
<?php

$manager

// Этому javascript расположен внизу страницы
->collection('jsFooter')
```

```
// Название получаемого файла
->setTargetPath('final.js')

// С таким URI генерируется мэг html
->setTargetUri('production/final.js')

// Это удаленный ресурс, не нуждающийся в фильтрации
->addJs('code.jquery.com/jquery-1.10.0.min.js', true, false)

// Это локальные ресурсы, к которым необходимо применить фильтры
->addJs('common-functions.js')
->addJs('page-functions.js')

// Объединяем все ресурсы в один файл
->join(true)

// Используем встроенный фильтр Jsmin
->addFilter(new Phalcon\Assets\Filters\Jsmin())

// Используем пользовательский фильтр
->addFilter(new MyApp\Assets\Filters\LicenseStamper());
```

Менеджер начинает получать набор ресурсов от Assets Manager, который может содержать либо javascript, либо CSS, но не оба типа ресурсов. Некоторые ресурсы могут быть удаленными, то есть, полученными с помощью HTTP запроса для дальнейшей фильтрации. Преобразования внешних ресурсов рекомендуется для устранения накладных расходов на их получение.

<?php

```
// Этот javascript расположен внизу
$js = $manager->collection('jsFooter');
```

Как показано выше, метод addJs используется для добавления ресурсов в коллекцию, второй параметр указывает, является ли ресурс внешним или нет, и третий параметр указывает, должен ли ресурс быть отфильтрован или нет:

<?php

```
// Это удаленный ресурс, не нуждающийся в фильтрации
$js->addJs('code.jquery.com/jquery-1.10.0.min.js', true, false);

// Это локальные ресурсы, к которым необходимо применить фильтры
$js->addJs('common-functions.js');
$js->addJs('page-functions.js');
```

Фильтры регистрируются в коллекции, допускается регистрировать несколько фильтров. Ресурсы в наборе фильтруются в том же порядке, в каком были зарегистрированы фильтры:

<?php

```
// Используем встроенный фильтр Jsmin
$js->addFilter(new Phalcon\Assets\Filters\Jsmin());

// Используем пользовательский фильтр
$js->addFilter(new MyApp\Assets\Filters\LicenseStamper());
```

Заметим, что встроенные и пользовательские фильтры могут сразу применяться к набору ресурсов. Последний шаг, определяет, стоит ли объединять все ресурсы набора в один файл, или использовать

каждый по отдельности. Если все ресурсы набора должны объединяться в один файл, вы можете использовать метод ‘join’:

```
<?php

// Объединяем все ресурсы в один файл
$js->join(true);

// Название получаемого файла
$js->setTargetPath('public/production/final.js');

// С таким URI генерируется тэг html
$js->setTargetUri('production/final.js');
```

Если ресурсы должны быть объединены, то вы должны также определить какой файл будет использоваться для хранения ресурсов и по какому URI он будет доступен. Эти параметры настраиваются с помощью методов setTargetPath() и setTargetUri().

## Встроенные фильтры

Phalcon имеет два встроенных фильтра минимизации javascript и CSS, их реализация на C обеспечивает минимальные накладные расходы для решения подобной задачи:

| Фильтр                              | Описание  |
|-------------------------------------|---|
| Phalcon\Assets\Filters\JsMinimizer  | Минимизирует JavaScript удаляя не нужны символы, которые игнорируются интерпретатором/компилятором JavaScript |
| Phalcon\Assets\Filters\CssMinimizer | Минимизирует CSS удаляя ненужные символы, которые игнорируются браузерами                                     |

## Пользовательские фильтры

Кроме использования встроенных фильтров, разработчик может создавать свои собственные фильтры. Вы можете воспользоваться существующими более продвинутыми инструментами, такими как YUI, Sass, Closure и другие.

```
<?php

use Phalcon\Assets\FilterInterface;

/**
 * Filters CSS content using YUI
 *
 * @param string $contents
 * @return string
 */
class CssYUICompressor implements FilterInterface
{

    protected $_options;

    /**
     * CssYUICompressor constructor
     *
     * @param array $options
     */
    public function __construct($options)
```

```

{
    $this->_options = $options;
}

/**
 * Do the filtering
 *
 * @param string $contents
 * @return string
 */
public function filter($contents)
{
    //Write the string contents into a temporal file
    file_put_contents('temp/my-temp-1.css', $contents);

    system(
        $this->_options['java-bin'] .
        '-jar ' .
        $this->_options['yui'] .
        '--type css ' .
        'temp/my-temp-file-1.css ' .
        $this->_options['extra-options'] .
        '-o temp/my-temp-file-2.css'
    );

    //Return the contents of file
    return file_get_contents("temp/my-temp-file-2.css");
}
}

```

Применение:

```

<?php

//Get some CSS collection
$css = $this->assets->get('head');

//Add/Enable the YUI compressor filter in the collection
$css->addFilter(new CssYUICompressor(array(
    'java-bin' => '/usr/local/bin/java',
    'yui' => '/some/path/yuicompressor-x.y.z.jar',
    'extra-options' => '--charset utf8'
)));

```

## 2.17.6 Пользовательский вывод

Методы outputJs и outputCss создают требуемую HTML-разметку в соответствии с каждым типом ресурсов, но вы можете переопределить эти методы и создать разметку вручную:

```

<?php

foreach ($this->assets->collection('js') as $resource) {
    echo \Phalcon\Tag::javascriptInclude($resource->getPath());
}

```

## 2.18 Шаблонизатор Volt

Volt — ультрабыстрый и дружелюбный по отношению к дизайнеру язык шаблонизации, написанный на С для PHP. Он предоставляет набор подручных средств, который позволит вам легко создавать представления. Volt очень сильно связан с остальными компонентами Phalcon, однако, вы можете использовать его в качестве самостоятельного компонента вашего приложения.

```

show.volt * show.volt UNREGISTERED
1  {# app/views/products/show.volt #}
2  {% extends "templates/products.volt" %}
3
4  {{ content() }}
5
6  {% block last_products %}
7
8  {% for product in products %}
9      * Name: {{ product.name|e }}
10     {% if product.status == "Active" %}
11         Price: {{ product.price + product.taxes/100 }}
12         {{ link_to('More', 'products/details/' ~ product.id) }}
13     {% else %}
14         {{ '(Inactive)' }}
15     {% endif %}
16 {% endfor %}
17
18 {% endblock %}
19
20 {% block footer %}
21 {{ partial('navbar/footer') }}
22 {% endblock %}

```

Line 1, Column 1 Tab Size: 4 HTML (Volt)

Volt был написан под вдохновлением от [Jinja](#), который был создан Armin Ronacher. По этой причине многие разработчики будут чувствовать себя как дома, используя такой же синтаксис, что и в похожих шаблонизаторах. Возможности и синтаксис Volt были улучшены многими вещами и, конечно же, производительностью, к которым так привыкли разработчики, работая с Phalcon.

### 2.18.1 Введение

Представления на Volt компилируются в чистый PHP код, избавляя тем самым от необходимости писать его вручную:

```

{# app/views/products/show.volt #-}

{% block last_products %}

{% for product in products %}
    * Name: {{ product.name|e }}
    {% if product.status == "Active" %}
        Price: {{ product.price + product.taxes/100 }}
    {% endif %}
{% endfor %}

{% endblock %}

```

## 2.18.2 Подключение Volt

Вы можете подключить Volt в компоненте представлений как любой другой шаблонизатор, используя при этом новое расширение для файлов, или всё то же стандартное .phtml:

```
<?php

//Registering Volt as template engine
$di->set('view', function() {

    $view = new \Phalcon\Mvc\View();

    $view->setViewsDir('../app/views/');

    $view->registerEngines(array(
        ".volt" => '\Phalcon\Mvc\View\Engine\Volt',
    ));

    return $view;
});
```

Использование стандартного расширения ".phtml":

```
<?php

$view->registerEngines(array(
    ".phtml" => '\Phalcon\Mvc\View\Engine\Volt',
));
```

## 2.18.3 Основы

Представление состоит из Volt кода, PHP и HTML. Набор специальных разделителей позволяет входить в режим Volt. Разделители { % ... % } используются для выполнения операторов, таких как циклы for и присваивания, а { { ... } } выводят результат выражения в шаблон.

Ниже представлен небольшой шаблон, иллюстрирующий эти основные возможности:

```
{# app/views/posts/show.phtml #}
<!DOCTYPE html>
<html>
    <head>
        <title>{{ title }} - A example blog</title>
    </head>
    <body>

        {% if show_navigation %}
            <ul id="navigation">
                {% for item in menu %}
                    <li><a href="{{ item.href }}">{{ item.caption }}</a></li>
                {% endfor %}
            </ul>
        {% endif %}

        <h1>{{ post.title }}</h1>

        <div class="content">
            {{ post.content }}
        </div>
```

```
</body>
</html>
```

Используя Phalcon\Mvc\View::setVar вы можете передать переменные из контроллера в представление. В предыдущем примере это были три переменные: title, menu и post:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function showAction()
    {

        $post = Post::findFirst();

        $this->view->setVar("title", $post->title);
        $this->view->setVar("post", $post);
        // или
        $this->view->menu = Menu::find();
        $this->view->show_navigation = true;

    }

}
```

#### 2.18.4 Переменные

Переменные могут иметь атрибуты, доступные при использовании синтаксиса: foo.bar. Если вы передаёте массивы, то обратиться к их элементам можно посредством квадратных скобок: foo['bar']

```
{{ post.title }}
{{ post['title'] }}
```

#### 2.18.5 Фильтры

Вывод переменных можно форматировать или модифицировать при помощи фильтров. Для их применения используется оператор | (вертикальная черта):

```
{{ post.title|e }}
{{ post.content|striptags }}
{{ name|capitalize|trim }}
```

Список встроенных в Volt фильтров:

| Фильтр           | Описание  |
|------------------|---|
| e                | Применяет к значению Phalcon\Escaper->escapeHtml  |
| escape           | Применяет к значению Phalcon\Escaper->escapeHtml  |
| escape_css       | Применяет к значению Phalcon\Escaper->escapeCss   |
| escape_js        | Применяет к значению Phalcon\Escaper->escapeJs  |
| escape_attr      | Применяет к значению Phalcon\Escaper->escapeHtmlAttr  |
| trim             | Применяет к значению PHP-функцию trim, которая удаляет лишние пробелы   |
| striptags        | Применяет к значению PHP-функцию strip_tags, удаляющую HTML тэги  |
| slashes          | Применяет к значению PHP-функцию addslashes, экранирующую значение  |
| stripslashes     | Применяет к значению PHP-функцию stripslashes, удаляющую экранирующие кавычки   |
| capitalize       | Делает первую букву строки заглавной, используя PHP-функцию ucwords   |
| lower            | Преобразует все символы строки к нижнему регистру   |
| upper            | Преобразует все символы строки к верхнему регистру  |
| length           | Подсчитывает длину строки, или количество элементов в массиве/объекте   |
| nl2br            | Изменяет \n на HTML вариант(<br />). Применяет функцию nl2br  |
| sort             | Sorts an array using the PHP function asort   |
| keys             | Возвращает ключи массива, используя array_keys  |
| join             | Объединяет части массива, используя join  |
| format           | Форматирует строку, используя sprintf.  |
| json_encode      | Преобразует значение в JSON с помощью функции json_encode   |
| json_decode      | Преобразует значение из JSON в PHP с помощью функции json_decode  |
| abs              | Применяет к значению PHP-функцию abs  |
| url_encode       | Применяет к значению PHP-функцию urlencode  |
| default          | Устанавливает значение по умолчанию, если полученное выражение пусто (переменная не задана, или содержит пустое значение) |
| convert_encoding | Преобразует строку из одной кодировки в другую  |

Примеры:

```

"># e или escape #
{{ "<h1>Hello<h1>"|e }}
{{ "<h1>Hello<h1>"|escape }}

# trim #
{{ "    hello    "|trim }}

# striptags #
{{ "<h1>Hello<h1>"|striptags }}

# slashes #
{{ "'this is a string'"|slashes }}

# stripslashes #
{{ "'this is a string\'|"|stripslashes }}

# capitalize #
{{ "hello"|capitalize }}

# lower #
{{ "HELLO"|lower }}

# upper #
{{ "hello"|upper }}

# length #
{{ "robots"|length }}

```

```
 {{ [1, 2, 3]|length }}

{# nl2br #}
{{ "some\ntext"|nl2br }}

{# sort filter #}
{% set sorted=[3, 1, 2]|sort %}

{# keys filter #}
{% set keys={'first': 1, 'second': 2, 'third': 3}|keys %}

{# json_encode filter #}
{{ robots|json_encode }}

{# json_decode filter #}
{% set decoded='{"one":1,"two":2,"three":3}'|json_decode %}

{# url_encode filter #}
{{ post.permanent_link|url_encode }}

{# convert_encoding filter #}
{{ "désolé"|convert_encoding('utf8', 'latin1') }}
```

## 2.18.6 Комментарии

В шаблон можно добавить комментарии, используя разделители `{# ... #}`. Любой текст внутри них будет проигнорирован и не попадёт в вывод:

```
{# note: this is a comment
  % set price = 100; %}
#}
```

## 2.18.7 Список управляющих конструкций

Volt позволяет использовать в шаблонах набор основных, но мощных управляющих структур:

### For

Цикл по всем элементам в последовательности. Пример ниже показывает, как пройти по набору “`robots`” и вывести их имена:

```
<h1>Robots</h1>
<ul>
  {% for robot in robots %}
    <li>{{ robot.name|e }}</li>
  {% endfor %}
</ul>
```

циклы так же могут быть вложенными:

```
<h1>Robots</h1>
{% for robot in robots %}
  {% for part in robot.parts %}
    Robot: {{ robot.name|e }} Part: {{ part.name|e }} <br/>
```

```
{% endfor %}
{% endfor %}
```

Вы можете получить ключи значений массива так же, как и в PHP используя такой синтаксис:

```
{% set numbers = ['one': 1, 'two': 2, 'three': 3] %}
{% for name, value in numbers %}
    Name: {{ name }} Value: {{ value }}
{% endfor %}
```

Кроме того для выборочного прохода по элементам, можно определить условие “if”:

```
{% set numbers = ['one': 1, 'two': 2, 'three': 3] %

{% for value in numbers if value < 2 %}
    Name: {{ name }} Value: {{ value }}
{% endfor %}

{% for name, value in numbers if name != 'two' %}
    Name: {{ name }} Value: {{ value }}
{% endfor %}
```

Если ‘else’ определяется внутри ‘for’, то этот блок будет выполнен в том случае, когда не будет произведено ни одной итерации:

```
<h1>Robots</h1>
{% for robot in robots %}
    Robot: {{ robot.name|e }} Part: {{ part.name|e }} <br/>
{% else %}
    There are no robots to show
{% endfor %}
```

Альтернативный синтаксис:

```
<h1>Robots</h1>
{% for robot in robots %}
    Robot: {{ robot.name|e }} Part: {{ part.name|e }} <br/>
{% elseif %}
    There are no robots to show
{% endfor %}
```

## Управление циклами

Такие операторы как ‘break’ and ‘continue’ могут быть использованы для выхода из цикла вообще, или перехода к следующей итерации:

```
{# пропустить робота с четным индексом #}
{% for index, robot in robots %}
    {% if index is even %}
        {% continue %}
    {% endif %}
    ...
{% endfor %}

{# выход из цикла при первом встреченном четном роботе #}
{% for index, robot in robots %}
    {% if index is even %}
```

```
    {%- break %}
{%- endif %}
...
{% endfor %}
```

## If

Как и в PHP оператор “if” проверяет значение выражения на ложь или истину:

```
<h1>Cyborg Robots</h1>
<ul>
{% for robot in robots %}
  {% if robot.type == "cyborg" %}
    <li>{{ robot.name|e }}</li>
  {% endif %}
{% endfor %}
</ul>
```

Условие else тоже поддерживается:

```
<h1>Robots</h1>
<ul>
{% for robot in robots %}
  {% if robot.type == "cyborg" %}
    <li>{{ robot.name|e }}</li>
  {% else %}
    <li>{{ robot.name|e }} (not a cyborg)</li>
  {% endif %}
{% endfor %}
</ul>
```

Структура “elseif” может быть использована совместно с “if” для повторения функционала “switch”:

```
{% if robot.type == "cyborg" %}
  Robot is a cyborg
{% elseif robot.type == "virtual" %}
  Robot is virtual
{% elseif robot.type == "mechanical" %}
  Robot is mechanical
{% endif %}
```

## Контекст цикла

Внутри цикла ‘for’ доступна специальная переменная, предоставляющая информацию о нём

| Переменная     | Описание   |
|----------------|--|
| loop.index     | Текущая итерация цикла (нумерация с 1)             |
| loop.index0    | Текущая итерация цикла (нумерация с 0)             |
| loop.revindex  | Номер итерации с конца цикла (нумерация с 1)       |
| loop.revindex0 | Номер итерации с конца цикла (нумерация с 0)       |
| loop.first     | Возвращает true, если текущая итерация — первая    |
| loop.last      | Возвращает true, если текущая итерация — последняя |
| loop.length    | Количество элементов для итерирования              |

```
{% for robot in robots %}
  {% if loop.first %}
```

```

<table>
<tr>
    <th>#</th>
    <th>Id</th>
    <th>Name</th>
</tr>
{%
    endif %}
<tr>
    <td>{{ loop.index }}</td>
    <td>{{ robot.id }}</td>
    <td>{{ robot.name }}</td>
</tr>
{%
    if loop.last %}
</table>
{%
    endif %}
{%
    endfor %}

```

## 2.18.8 Присваивания

Переменные могут быть изменены в шаблоне. для этого используется оператор “set”:

```

{%
    set fruits = ['Apple', 'Banana', 'Orange'] %}
{%
    set name = robot.name %}

```

Multiple assignments are allowed in the same instruction:

```
{%
    set fruits = ['Apple', 'Banana', 'Orange'], name = robot.name, active = true %}
```

Additionally, you can use compound assignment operators:

```

{%
    set price += 100.00 %}
{%
    set age *= 5 %}

```

The following operators are available:

| Operator | Description               |
|----------|---------------------------|
| =        | Standard Assignment       |
| +=       | Addition assignment       |
| -=       | Subtraction assignment    |
| *=       | Multiplication assignment |
| /=       | Division assignment       |

## 2.18.9 Выражения

Volt позволяет использовать базовый набор выражений, включая литералы.

Выражения вычисляются и выводятся с использованием разделителей ‘{{‘ и ‘}}’:

```
{{ (1 + 1) * 2 }}
```

If an expression needs to be evaluated without be printed the ‘do’ statement can be used:

```
{%
    do (1 + 1) * 2 %}
```

## Литералы

Поддерживаются следующие литералы:

| Литералы     | Описание   |
|--------------|--|
| “это строка” | Текст, заключенный в двойные или одинарные кавычки воспринимается как строка |
| 100.25       | Числа, с десятичной частью воспринимаются как числа с плавающей запятой      |
| 100          | Числа без десятичной части воспринимаются как целые                          |
| false        | Константа “false” воспринимается как булевое значение “false”                |
| true         | Константа “true” воспринимается как булевое значение “true”                  |
| null         | Константа “null” воспринимается как NULL-значение                            |

## Массивы

Если вы используете PHP 5.3 or 5.4, 5.5, то можете создавать массивы, перечисляя список значений в квадратных скобках:

```
{# Простой массив #}
{{ ['Apple', 'Banana', 'Orange'] }}

{# Еще один простой массив #}
{{ ['Apple', 1, 2.5, false, null] }}

{# Многомерный массив #}
{{ [[1, 2], [3, 4], [5, 6]] }}

{# Хеш-массив #}
{{ ['first': 1, 'second': 4/2, 'third': '3'] }}
```

Curly braces also can be used to define arrays or hashes:

```
% set myArray = {'Apple', 'Banana', 'Orange'} %
% set myHash = {'first': 1, 'second': 4/2, 'third': '3'} %
```

## Математические операторы

Вы можете производить вычисления в шаблонах, используя следующие операторы:

| Оператор | Оператор  |
|----------|---|
| +        | Производит операцию сложения. {{ 2 + 3 }} вернёт 5              |
| -        | Производит операцию вычитания. {{ 2 - 3 }} вернёт -1            |
| *        | Производит операцию умножения. {{ 2 * 3 }} вернёт 6             |
| /        | Производит операцию деления. {{ 10 / 2 }} вернёт 5              |
| %        | Вычисляет остаток от деления целых чисел. {{ 10 % 3 }} вернёт 1 |

## Операторы сравнения

Доступны следующие операторы сравнения:

| Оператор              | Описание  |
|-----------------------|---|
| <code>==</code>       | Проверяет равенство двух операндов                    |
| <code>!=</code>       | Проверяет неравенство двух операндов                  |
| <code>&lt;&gt;</code> | Проверяет неравенство двух операндов                  |
| <code>&gt;</code>     | Проверяет, что левый операнд больше, чем правый       |
| <code>&lt;</code>     | Проверяет, что левый операнд меньше, чем правый       |
| <code>&lt;=</code>    | Проверяет, что левый операнд меньше или равен правому |
| <code>&gt;=</code>    | Проверяет, что левый операнд больше или равен правому |
| <code>====</code>     | Проверяет строгое равенство операндов                 |
| <code>!==</code>      | Проверяет строгое неравенство операндов               |

## Логические операторы

Логические операторы полезны в выражении “if” чтобы объединить несколько проверок:

| Оператор                   | Описание  |
|----------------------------|---|
| <code>or</code>            | Возвращает true, если левый или правый операнды возвращают true               |
| <code>and</code>           | Возвращает true, если одновременно и левый, и правый операнды возвращают true |
| <code>not</code>           | Отрицание выражения   |
| <code>( выражение )</code> | Скобки для группирования выражений  |

## Другие операторы

Доступны так же дополнительные операторы:

| Оператор                     | Описание   |
|------------------------------|--|
| <code>~</code>               | Конкатенация двух операндов <code>{{ "hello" ~ "world" }}</code>                           |
| <code> </code>               | Применяет фильтр, указанный справа к операнду слева <code>{{ "hello"   uppercase }}</code> |
| <code>..</code>              | Создаёт диапазон значений <code>{{ 'a'..'z' }}</code> <code>{{ 1..10 }}</code>             |
| <code>is</code>              | То же самое, что и <code>==</code> (равно), также выполняет проверки (см. ниже)            |
| <code>in</code>              | Проверяет, что выражение содержится в другом выражении <code>if "a" in "abc"</code>        |
| <code>is not</code>          | То же самое, что и <code>!=</code> (не равно)  |
| <code>'a' ? 'b' : 'c'</code> | Тернарный оператор. Аналогичен тернарному оператору в PHP                                  |
| <code>++</code>              | Increments a value   |
| <code>-</code>               | Decrements a value   |

Пример ниже показывает их использование:

```
{% set robots = ['Voltron', 'Astro Boy', 'Terminator', 'C3PO'] %}

{% for index in 0..robots|length %}
    {% if robots[index] is defined %}
        {{ "Name: " ~ robots[index] }}
    {% endif %}
{% endfor %}
```

### 2.18.10 Проверки

Проверки могут быть использованы для определения соответствия переменной какому-то ожидаемому значению. Оператор “is” используется для выполнения проверок:

```
{% set robots = [1: 'Voltron', 2: 'Astro Boy', 3: 'Terminator', 4: 'C3PO'] %}

{% for position, name in robots %}
    {% if position is odd %}
        {{ value }}
    {% endif %}
{% endfor %}
```

The following built-in tests are available in Volt:

| Проверка    | Описание  |
|-------------|---|
| defined     | Проверяет существование переменной (isset)  |
| empty       | Проверяет, если значение пусто  |
| even        | Проверяет чётность целочисленного значения  |
| odd         | Проверяет нечётность целочисленного значения  |
| numeric     | Проверяет, является ли значение числом  |
| scalar      | Проверяет, что значение скаляр (не массив или объект)                                   |
| iterable    | Проверяет, является ли значение итерируемым, т.е. может быть использовано в цикле "for" |
| divisibleby | Проверяет, делится ли значение на другое без остатка                                    |
| sameas      | Проверяет, что значение совпадает с другим  |
| type        | Проверяет специфичный тип переменной  |

Больше примеров:

```
{% if robot is defined %}
    The robot variable is defined
{% endif %}

{% if robot is empty %}
    The robot is null or isn't defined
{% endif %}

{% for key, name in [1: 'Voltron', 2: 'Astroy Boy', 3: 'Bender'] %}
    {% if key is even %}
        {{ name }}
    {% endif %}
{% endfor %}

{% for key, name in [1: 'Voltron', 2: 'Astroy Boy', 3: 'Bender'] %}
    {% if key is odd %}
        {{ name }}
    {% endif %}
{% endfor %}

{% for key, name in [1: 'Voltron', 2: 'Astroy Boy', 'third': 'Bender'] %}
    {% if key is numeric %}
        {{ name }}
    {% endif %}
{% endfor %}

{% set robots = [1: 'Voltron', 2: 'Astroy Boy'] %}
{% if robots is iterable %}
    {% for robot in robots %}
        ...
    {% endfor %}
{% endif %}
```

```

{%- set world = "hello" %}
{%- if world is sameas("hello") %}
    {{ "it's hello" }}
{%- endif %}

{%- set external = false %}
{%- if external is type('boolean') %}
    {{ "external is false or true" }}
{%- endif %}

```

## 2.18.11 Macros

Макросы могут быть использованы для избежания повторений в шаблоне, они действуют как функции PHP, они могут получать параметры и возвращать значения:

```

<# Макрос "Вывода списка ссылок на похожие темы" #>
{%- macro related_bar(related_links) %}
    <ul>
        {%- for link in related_links %}
            <li><a href="{{ url(link.url) }}" title="{{ link.title|striptags }}">{{ link.text }}</a></li>
        {%- endfor %}
    </ul>
{%- endmacro %}

<# Используем макрос "Вывода списка ссылок на похожие темы" #>
{{ related_bar(links) }}

<div>This is the content</div>

<# Используем макрос "Вывода списка ссылок на похожие темы" снова #>
{{ related_bar(links) }}

```

При использовании макросов, параметры могут быть переданы по имени:

```

{%- macro error_messages(message, field, type) %}
    <div>
        <span class="error-type">{{ type }}</span>
        <span class="error-field">{{ field }}</span>
        <span class="error-message">{{ message }}</span>
    </div>
{%- endmacro %}

<# Использование макроса #>
{{ error_messages('type': 'Invalid', 'message': 'The name is invalid', 'field': 'name') }}

```

Макросы могут возвращать значения:

```

{%- macro my_input(name, class) %}
    {{ return text_field(name, 'class': class) }}
{%- endmacro %}

<# Использование макроса #>
{{ '<p>' ~ my_input('name', 'input-text') ~ '</p>' }}

```

И задавать параметры по умолчанию:

```
{%- macro my_input(name, class="input-text") %}
  {% return text_field(name, 'class': class) %}
{%- endmacro %}

{# Использование макрояда #}
{{ '<p>' ~ my_input('name') ~ '</p>' }}
{{ '<p>' ~ my_input('name', 'input-text') ~ '</p>' }}
```

## 2.18.12 Использование Tag Helpers

Volt сильно связан с *Phalcon Tag*, поэтому можно легко использовать в Volt-шаблонах helpers, предоставляемые этим компонентом:

```
{{ javascript_include("js/jquery.js") }}

{{ form('products/save', 'method': 'post') }}

<label>Name</label>
{{ text_field("name", "size": 32) }}

<label>Type</label>
{{ select("type", productTypes, 'using': ['id', 'name']) }}

{{ submit_button('Send') }}

</form>
```

В результате будет сгенерирован следующий PHP-код:

```
<?php echo Phalcon\Tag::javascriptInclude("js/jquery.js") ?>

<?php echo Phalcon\Tag::form(array('products/save', 'method' => 'post')) ; ?>

<label>Name</label>
<?php echo Phalcon\Tag::textField(array('name', 'size' => 32)); ?>

<label>Type</label>
<?php echo Phalcon\Tag::select(array('type', $productTypes, 'using' => array('id', 'name'))); ?>

<?php echo Phalcon\Tag::submitButton('Send'); ?>

</form>
```

Для вызова PhalconTag helper, вам необходимо лишь вызвать соответствующие версии методов не в Camelcase:

Метод	Функция Volt
Phalcon\Tag::linkTo	link_to
Phalcon\Tag::textField	text_field
Phalcon\Tag::passwordField	password_field
Phalcon\Tag::hiddenField	hidden_field
Phalcon\Tag::fileField	file_field
Phalcon\Tag::checkField	check_field
Phalcon\Tag::radioField	radio_field
Phalcon\Tag::dateField	date_field
Phalcon\Tag::emailField	email_field
Phalcon\Tag::numberField	number_field
Phalcon\Tag::submitButton	submit_button
Phalcon\Tag::selectStatic	select_static
Phalcon\Tag::select	select
Phalcon\Tag::textArea	text_area
Phalcon\Tag::form	form
Phalcon\Tag::endForm	end_form
Phalcon\Tag::getTitle	get_title
Phalcon\Tag::stylesheetLink	stylesheet_link
Phalcon\Tag::javascriptInclude	javascript_include
Phalcon\Tag::image	image
Phalcon\Tag::friendlyTitle	friendly_title

### 2.18.13 Функции

В Volt доступны перечисленные ниже встроенные функции:

Название	Описание
content	Включает результат рендера предыдущего этапа
get_content	То же самое, что и ‘content’
partial	Динамически загружает partial представление в текущий шаблон
super	Отрисовывает содержимое родительского блока
time	Вызывает одноимённую PHP-функцию
date	Вызывает одноимённую PHP-функцию
dump	Вызывает PHP-функцию ‘var_dump’
version	Возвращает текущую версию фреймворка
constant	Читает PHP константу
url	Генерирует URL, используя сервис ‘url’

### 2.18.14 Связывание с представлениями

Кроме того, Volt связан с *Phalcon Mvc View*, что позволяет вам поиграться с иерархией и включением partials:

```
{{ content() }}

<!-- Simple include of a partial -->
<div id="footer">{{ partial("partials/footer") }}</div>

<!-- Passing extra variables -->
<div id="footer">{{ partial("partials/footer", ['links': $links]) }}</div>
```

Partial включается в момент выполнения, Volt так же предоставляет “include”, которая собирает содержимое представления и возвращает его в виде включаемой части:

```
{# Simple include of a partial #}
<div id="footer">{% include "partials/footer" %}</div>

{# Passing extra variables #}
<div id="footer">{%
    include "partials/footer"
    with [ 'links' : links ]
%}</div>
```

## Include

‘include’ has a special behavior that will help us improve performance a bit when using Volt, if you specify the extension when including the file and it exists when the template is compiled, Volt can inline the contents of the template in the parent template where it’s included. Templates aren’t inlined if the ‘include’ have variables passed with ‘with’:

```
{# The contents of 'partials/footer.volt' is compiled and inlined #}
<div id="footer">{%
    include "partials/footer.volt"
%}</div>
```

### 2.18.15 Наследование шаблонов

С помощью наследования шаблонов вы можете создавать базовые шаблоны, которые могут быть расширены другими шаблонами, что позволит повторно использовать уже написанный код. Базовый шаблон определяет *блоки*, которые могут быть переопределены дочерними шаблонами. Предположим, что у нас есть некоторый базовый шаблон:

```
{# templates/base.volt #}
<!DOCTYPE html>
<html>
    <head>
        {% block head %}
            <link rel="stylesheet" href="style.css" />
        {% endblock %}
        <title>{% block title %}{% endblock %} - My Webpage</title>
    </head>
    <body>
        <div id="content">{% block content %}{% endblock %}</div>
        <div id="footer">
            {% block footer %}&copy; Copyright 2012, All rights reserved.{% endblock %}
        </div>
    </body>
</html>
```

Заменив блоки, мы расширим базовый шаблон другим:

```
{% extends "templates/base.volt" %}

{% block title %}Index{% endblock %}

{% block head %}<style type="text/css">.important { color: #336699; }</style>{% endblock %}

{% block content %}
    <h1>Index</h1>
    <p class="important">Welcome on my awesome homepage.</p>
{% endblock %}
```

Не обязательно заменять все блоки дочерними шаблонами, можно только те, которые необходимо. В результате, вывод будет таким:

```
<!DOCTYPE html>
<html>
    <head>
        <style type="text/css">.important { color: #336699; }</style>
        <title>Index - My Webpage</title>
    </head>
    <body>
        <div id="content">
            <h1>Index</h1>
            <p class="important">Welcome on my awesome homepage.</p>
        </div>
        <div id="footer">
            &copy; Copyright 2012, All rights reserved.
        </div>
    </body>
</html>
```

## Множественное наследование

Шаблоны, которые наследуют другие шаблоны, так же могут быть унаследованы. Это иллюстрирует следующий пример:

```
{# main.volt #}
<!DOCTYPE html>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>
        {% block content %}{{ endblock %}}
    </body>
</html>
```

Шаблон “layout.volt” наследует “main.volt”

```
{# layout.volt #}
{% extends "main.volt" %}

{% block content %}

    <h1>Table of contents</h1>

{% endblock %}
```

Финальное представление, наследуемое “layout.volt”:

```
{# index.volt #}
{% extends "layout.volt" %}

{% block content %}

    {{ super() }}

    <ul>
        <li>Some option</li>
        <li>Some other option</li>
    </ul>
```

```
{% endblock %}
```

Отрисовка “index.volt”:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Title</title>
    </head>
    <body>

        <h1>Table of contents</h1>

        <ul>
            <li>Some option</li>
            <li>Some other option</li>
        </ul>

    </body>
</html>
```

Обратите внимание на вызов функции “super()”. Эта функция позволяет отрисовать содержимое родительского блока.

Как и partials, путь, установленный в “extends” — это путь относительно текущей папки с представлениями (т.е. app/views/).

По умолчанию и из соображений производительности, Volt проверяет только изменения в дочерних шаблонах, чтобы понять, когда нужно снова пересобрать PHP, поэтому рекомендуется инициализировать Volt с опцией ‘compileAlways’ => true. Таким образом, шаблоны компилируются с учётом изменений родительского шаблона.

## 2.18.16 Режим автоматического экранирования

Вы можете включить режим автоматического экранирования всех выводимых в блоке переменных:

```
Manually escaped: {{ robot.name|e }}

{% autoescape true %}
    Autoescaped: {{ robot.name }}
    {% autoescape false %}
        No Autoescaped: {{ robot.name }}
    {% endautoescape %}
{% endautoescape %}
```

## 2.18.17 Настройка шаблонизатора Volt

Volt можно настроить так, чтобы изменить его поведение по умолчанию. В следующем примере объясняется, как это можно сделать:

```
<?php

use Phalcon\Mvc\View,
    Phalcon\Mvc\View\Engine\Volt;

//Register Volt as a service
```

```

$di->set('voltService', function($view, $di) {
    $volt = new Volt($view, $di);

    $volt->setOptions(array(
        "compiledPath" => "../app/compiled-templates/",
        "compiledExtension" => ".compiled"
    ));

    return $volt;
});

//Register Volt as template engine
$di->set('view', function() {

    $view = new View();

    $view->setViewsDir('../app/views/');

    $view->registerEngines(array(
        ".volt" => 'voltService'
    ));

    return $view;
});

```

Если вы не хотите использовать Volt в качестве сервиса, вы можете передать при регистрации шаблонизатора анонимную функцию, вместо имени сервиса:

```

<?php

// Регистрация Volt в качестве шаблонизатора с анонимной функцией
$di->set('view', function() {

    $view = new \Phalcon\Mvc\View();

    $view->setViewsDir('../app/views/');

    $view->registerEngines(array(
        ".volt" => function($view, $di) {
            $volt = new \Phalcon\Mvc\View\Engine\Volt($view, $di);

            // тут установка каких-то настроек

            return $volt;
        }
    ));

    return $view;
});

```

В Volt могут быть следующие опции:

Опция	Описание	По умолчанию
compiledPath	Путь для записи скомпилированных шаблонов	./
compiledExtension	Дополнительное расширение, добавляемое к скомпилированным PHP-файлам	.php
compiledSeparator	Volt заменяет разделители папок / и \ этим разделителем для создания одного файла в папке скомпилированных PHP файлов	%%
stat	Если Phalcon должен проверять, существуют ли различия между файлом шаблона и его скомпилированным результатом	true
compileAlways	Указывает Volt, должны ли шаблоны собираться на каждый запрос, или только тогда, когда они изменяются	false
prefix	Позволяет добавлять префикс к шаблонам в папке скомпилированных PHP файлов	null

The compilation path is generated according to the above options, if the developer wants total freedom defining the compilation path, an anonymous function can be used to generate the compilation path, this function receives the relative path to the template in the views directory. The following examples show how to change the compilation path dynamically:

```
<?php

// Just append the .php extension to the template path
// leaving the compiled templates in the same directory
$volt->setOptions(array(
    'compiledPath' => function($templatePath) {
        return $templatePath . '.php';
    }
));

// Recursively create the same structure in another directory
$volt->setOptions(array(
    'compiledPath' => function($templatePath) {
        $dirName = dirname($templatePath);
        if (!is_dir('cache/' . $dirName)) {
            mkdir('cache/' . $dirName);
        }
        return 'cache/' . $dirName . '/' . $templatePath . '.php';
    }
));
```

## 2.18.18 Расширение Volt

В отличие от других шаблонизаторов, Volt не требуется для запуска скомпилированных шаблонов. После того, как шаблон был собран, он больше никак не зависит от Volt. Иными словами, он используется лишь в качестве компилятора для PHP-шаблонов.

Volt-компилятор позволяет вам расширить его, добавив больше функций, проверки или фильтр к уже существующим.

### Функции

Функции действуют как обычные PHP-функции, поэтому им требуется строковое имя, разрешенное для функций в PHP. Функции можно добавить двумя способами: передать простое строчное имя, или использовать анонимную функцию. Любой способ должен возвращать допустимое PHP-выражение.

```
<?php

$volt = new \Phalcon\Mvc\View\Engine\Volt($view, $di);

$compiler = $volt->getCompiler();

// Тут к функции 'shuffle' в Volt призывается PHP-функция 'str_shuffle'
$compiler->addFunction('shuffle', 'str_shuffle');
```

При регистрации функции, как анонимной, мы используем `$resolvedArgs` для передачи аргументов точно так же, как они были приняты:

```
<?php

$compiler->addFunction('widget', function($resolvedArgs, $exprArgs) {
    return 'MyLibrary\Widgets::get('.implode(',',$resolvedArgs.'));
});
```

Учитывайте, что параметры независимы или не переданы:

```
<?php

$compiler->addFunction('repeat', function($resolvedArgs, $exprArgs) use ($compiler) {

    // Получение первого параметра
    $firstArgument = $compiler->expression($exprArgs[0]['expr']);

    // Проверка, что второй параметр был передан
    if (isset($exprArgs[1])) {
        $secondArgument = $compiler->expression($exprArgs[1]['expr']);
    } else {
        // По умолчанию используется '10'
        $secondArgument = '10';
    }

    return 'str_repeat(' . $firstArgument . ', ' . $secondArgument . ')';
});
```

Генерация кода на основе некоторой готовой функции:

```
<?php

$compiler->addFunction('contains_text', function($resolvedArgs, $exprArgs) {
    if (function_exists('mb_stripos')) {
        return 'mb_stripos(' . $resolvedArgs . ')';
    } else {
        return 'stripos(' . $resolvedArgs . ')';
    }
});
```

Встроенные функции могут быть перегружены добавлением функций с таким же именем:

```
<?php

// Заменяет встроенную функцию 'dump'
$compiler->addFunction('dump', 'print_r');
```

## Фильтры

Фильтры имеют следующий вид в шаблоне: leftExpr|name(optional-args). Добавление новых фильтров аналогично добавлению функций:

```
<?php  
  
// Создаёт фильтр 'hash', который использует функцию PHP 'md5'  
$compiler->addFilter('hash', 'md5');
```

```
<?php  
  
$compiler->addFilter('int', function($resolvedArgs, $exprArgs) {  
    return 'intval(' . $resolvedArgs . ')';  
});
```

Встроенные фильтры могут быть перегружены добавлением фильтра с таким же именем:

```
<?php  
  
//Replace built-in filter 'capitalize'  
$compiler->addFilter('capitalize', 'lcfirst');
```

## Extensions

With extensions the developer has more flexibility to extend the template engine, and override the compilation of a specific instruction, change the behavior of an expression or operator, add functions/filters, and more.

An extension is a class that implements the events triggered by Volt as a method of itself.

For example, the class below allows to use any PHP function in Volt:

```
<?php  
  
class PhpFunctionExtension  
{  
    /**  
     * This method is called on any attempt to compile a function call  
     */  
    public function compileFunction($name, $arguments)  
    {  
        if (function_exists($name)) {  
            return $name . '(' . $arguments . ')';  
        }  
    }  
}
```

The above class implements the method ‘compileFunction’ which is invoked before any attempt to compile a function call in any template. The purpose of the extension is to verify if a function to be compiled is a PHP function allowing to call it from the template. Events in extensions must return valid PHP code, this will be used as result of the compilation instead of the one generated by Volt. If an event doesn’t return a string the compilation is done using the default behavior provided by the engine.

The following compilation events are available to be implemented in extensions:

Event/Method	Description
compileFunction	Triggered before trying to compile any function call in a template
compileFilter	Triggered before trying to compile any filter call in a template
resolveExpression	Triggered before trying to compile any expression. This allows the developer to override operators
compileStatement	Triggered before trying to compile any expression. This allows the developer to override any statement

Volt extensions must be registered in the compiler making them available in compile time:

```
<?php
//Register the extension in the compiler
$compiler->addExtension(new PhpFunctionExtension());
```

## 2.18.19 Кэширование частей представления

С помощью Volt легко можно кэшировать части представления. Это повышает производительность, предотвращая выполнение PHP содержимого блока каждый раз, когда он отображается:

```
{% cache "sidebar" %}
    <!-- generate this content is slow so we are going to cache it --&gt;
{% endcache %}</pre>

```

Установка времени жизни кэша на определённое количество секунд:

```
{# кэширование сайдбара на 1 час #}
{% cache "sidebar" 3600 %}
    <!-- генерация этого содержимого достаточно медленна и мы решили её закэшировать --&gt;
{% endcache %}</pre>

```

В качестве ключа кэша может быть использовано любое разрешённое выражение:

```
{% cache ("article-" ~ post.id) 3600 %}

<h1>{{ post.title }}</h1>

<p>{{ post.content }}</p>

{% endcache %}
```

Кэширование выполняется компонентом *Phalcon\|Cache* через компонент представления. Узнать больше о том, как это работает можно в разделе “Caching View Fragments”.

## 2.18.20 Использование сервисов в шаблоне

Если контейнер сервисов (DI) доступен для Volt, вы можете использовать сервисы в шаблоне, получая доступ к ним по их именам:

```
{# Использование сервиса 'flash' #}
<div id="messages">{{ flash.output() }}</div>

{# Использование сервиса 'security' #}
<input type="hidden" name="token" value="{{ security.getToken() }}">
```

## 2.18.21 Отдельный компонент

Ниже продемонстрировано использование Volt, как отдельного компонента:

```
<?php

// Создание компилятора
$compiler = new \Phalcon\Mvc\View\Engine\Volt\Compiler();

// Добавление каких-то опций
$compiler->setOptions(array(
    ...
));

// Компиляция шаблона-строки, возвращающая PHP-код
echo $compiler->compileString('{{ "hello" }}');

// Компиляция шаблона-файла в определённый файл
$compiler->compileFile('layouts/main.volt', 'cache/layouts/main.volt.php');

// Компиляция шаблона-файла, в файл, определённый в настройках, переданных в компилятор
$compiler->compile('layouts/main.volt');

// Запрос собранных шаблонов (по желанию)
require $compiler->getCompiledTemplatePath();
```

## 2.18.22 Внешние ресурсы

- Пакет для Sublime/Textmate можно скачать [[на Github](#)]
- **Album-O-Rama** — пример приложения, использующего Volt в качестве шаблонизатора, [[код album-o-rama на Github](#)]
- Наш сайт работает на шаблонизаторе Volt, [[код website на Github](#)]
- **Phosphorum**, форум Phalcon так же использует Volt, [[код forum на Github](#)]
- **Vökuró**, еще одно приложение с использованием Volt, [[код vokuro на Github](#)]

## 2.19 MVC Приложения

Всю тяжелую работу при планировании работы MVC в Phalcon обычно выполняет **Phalcon\ Mvc\ Application**. Этот компонент инкапсулирует все сложные задачи необходимые изнутри, создаёт компоненты и интегрирует их в проект для реализации шаблона MVC по своему желанию.

### 2.19.1 Одномодульные и мультимодульные приложения

С помощью этого компонента можно запускать разные типы MVC приложений.

#### Одномодульные

Одномодульное MVC приложение состоит лишь из одного модуля. Можно использовать пространства имён, но необязательно. Такое приложение может иметь такую структуру:

```
single/
app/
    controllers/
    models/
    views/
public/
    css/
    img/
    js/
```

Если не используется пространство имён, то в качестве файла загрузки MVC можно использовать:

```
<?php

use Phalcon\Loader,
    Phalcon\DI\FactoryDefault,
    Phalcon\Mvc\Application,
    Phalcon\Mvc\View;

$loader = new Loader();

$loader->registerDirs(
    array(
        '../apps/controllers/',
        '../apps/models/'
    )
)->register();

$di = new FactoryDefault();

// Регистрация компонента представлений
$di->set('view', function() {
    $view = new View();
    $view->setViewsDir('../apps/views/');
    return $view;
});

try {
    $application = new Application($di);

    echo $application->handle()->getContent();

} catch (\Exception $e) {
    echo $e->getMessage();
}
```

Если же используются пространства имён, то файл может быть таким:

```
<?php

use Phalcon\Loader,
    Phalcon\Mvc\View,
    Phalcon\DI\FactoryDefault,
    Phalcon\Mvc\Dispatcher,
    Phalcon\Mvc\Application;

$loader = new Loader();
```

```
// Использование автозагрузки по префиксу пространства имен
$loader->registerNamespaces(
    array(
        'Single\Controllers' => '../apps/controllers/',
        'Single\Models'      => '../apps/models/',
    )
)->register();

$di = new FactoryDefault();

// Регистрация диспетчера с пространством имен для контроллеров
$di->set('dispatcher', function() {
    $dispatcher = new Dispatcher();
    $dispatcher->setDefaultNamespace('Single\Controllers');
    return $dispatcher;
});

// Регистрация компонента представлений
$di->set('view', function() {
    $view = new View();
    $view->setViewsDir('../apps/views/');
    return $view;
});

try {

    $application = new Application($di);
    echo $application->handle()->getContent();

} catch(\Exception $e){
    echo $e->getMessage();
}
```

## Мультимодульные

Мультимодульное приложение использует единый корень документов для нескольких модулей приложения. Файловая структура тогда может быть такой:

```
multiple/
  apps/
    frontend/
      controllers/
      models/
      views/
      Module.php
    backend/
      controllers/
      models/
      views/
      Module.php
  public/
    css/
    img/
    js/
```

Каждый каталог в `apps/` содержит собственную MVC структуру. Файл `Module.php` внутри такого каталога создан для настройки параметров этого модуля, таких как автозагрузка и настраиваемые

сервисы.

```
<?php

namespace Multiple\Backend;

use Phalcon\Loader,
    Phalcon\Mvc\Dispatcher,
    Phalcon\Mvc\View,
    Phalcon\Mvc\ModuleDefinitionInterface;

class Module implements ModuleDefinitionInterface
{

    /**
     * Регистрация автозагрузчика, специфичного для текущего модуля
     */
    public function registerAutoloaders()
    {

        $loader = new Loader();

        $loader->registerNamespaces(
            array(
                'Multiple\Backend\Controllers' => '../apps/backend/controllers/',
                'Multiple\Backend\Models'      => '../apps/backend/models/',
            )
        );

        $loader->register();
    }

    /**
     * Регистрация специфичных сервисов для модуля
     */
    public function registerServices($di)
    {

        // Регистрация диспетчера
        $di->set('dispatcher', function() {
            $dispatcher = new Dispatcher();
            $dispatcher->setDefaultNamespace("Multiple\Backend\Controllers\\");
            return $dispatcher;
        });

        // Регистрация компонента представлений
        $di->set('view', function() {
            $view = new View();
            $view->setViewsDir('../apps/backend/views/');
            return $view;
        });
    }
}
```

Для загрузки мультимодульных MVC приложений можно использовать такой файл автозагрузки:

```
<?php
```

```
use Phalcon\Mvc\Router,
    Phalcon\Mvc\Application,
    Phalcon\DI\FactoryDefault;

$di = new FactoryDefault();

// Специфичные роуты для модуля
$di->set('router', function () {

    $router = new Router();

    $router->setDefaultModule("frontend");

    $router->add("/login", array(
        'module'      => 'backend',
        'controller'  => 'login',
        'action'       => 'index',
    ));

    $router->add("/admin/products/:action", array(
        'module'      => 'backend',
        'controller'  => 'products',
        'action'       => 1,
    ));

    $router->add("/products/:action", array(
        'controller'  => 'products',
        'action'       => 1,
    ));

    return $router;
});

try {

    // Создание приложения
    $application = new Application($di);

    // Регистрация установленных модулей
    $application->registerModules(
        array(
            'frontend' => array(
                'className' => 'Multiple\Frontend\Module',
                'path'       => '../apps/frontend/Module.php',
            ),
            'backend'   => array(
                'className' => 'Multiple\Backend\Module',
                'path'       => '../apps/backend/Module.php',
            )
        )
    );

    // Обработка запроса
    echo $application->handle()->getContent();

} catch(\Exception $e){
    echo $e->getMessage();
}
```

Если вы хотите разместить в файле загрузки модуль с конфигурацией, вы можете использовать анонимную функцию для его регистрации:

```
<?php

// Создание компонента представлений
$view = new \Phalcon\Mvc\View();

// Установка параметров компонента представлений
//...

// Регистрация установленных модулей
$application->registerModules(
    array(
        'frontend' => function($di) use ($view) {
            $di->setShared('view', function() use ($view) {
                $view->setViewsDir('../apps/frontend/views/');
                return $view;
            });
        },
        'backend' => function($di) use ($view) {
            $di->setShared('view', function() use ($view) {
                $view->setViewsDir('../apps/backend/views/');
                return $view;
            });
        }
    )
);
```

Когда *Phalcon\ Mvc\ Application* зарегистрирует модули, всегда необходимо чтобы каждая регистрация возвращала существующий модуль. Каждый зарегистрированный модуль должен иметь соответствующий класс и функцию для настройки самого модуля. Каждый модуль должен обязательно содержать два метода: registerAutoloaders() и registerServices(), они будут автоматически вызваны *Phalcon\ Mvc\ Application* при выполнении модуля.

## 2.19.2 Понятие поведения по умолчанию

Если вы смотрели *руководство* или сгенерировали код используя *Инструменты разработчика*, вы можете узнать следующий код:

```
<?php

try {

    // Регистрация автозагрузчика
    //...

    // Регистрация сервисов
    //...

    // Обработка запроса
    $application = new \Phalcon\Mvc\Application($di);

    echo $application->handle()->getContent();

} catch (\Exception $e) {
    echo "Exception: ", $e->getMessage();
```

```
}
```

Ядро выполняет основную работу по запуску контроллера, при вызове handle():

```
<?php  
echo $application->handle()->getContent();
```

### 2.19.3 Ручная начальная загрузка

Если вы не хотите использовать *Phalcon\ Mvc\ Application*, код выше можно изменить вот так:

```
<?php  
  
// Получаем сервис из контейнера сервисов  
$router = $di['router'];  
  
$router->handle();  
  
$view = $di['view'];  
  
$dispatcher = $di['dispatcher'];  
  
// Передаём обработанные параметры маршрутизатора в диспетчер  
$dispatcher->setControllerName($router->getControllerName());  
$dispatcher->setActionName($router->getActionName());  
$dispatcher->setParams($router->getParams());  
  
// Запускаем представление  
$view->start();  
  
// Выполняем запрос  
$dispatcher->dispatch();  
  
// Выводим необходимое представление  
$view->render(  
    $dispatcher->getControllerName(),  
    $dispatcher->getActionName(),  
    $dispatcher->getParams()  
);  
  
// Завершаем работу представления  
$view->finish();  
  
$response = $di['response'];  
  
// Передаём результатом для ответа  
$response->setContent($view->getContent());  
  
// Отправляем заголовки  
$response->sendHeaders();  
  
// Выводим ответ  
echo $response->getContent();
```

The following replacement of *Phalcon\ Mvc\ Application* lacks of a view component making it suitable for Rest APIs:

```
<?php

// Get the 'router' service
$router = $di['router'];

$router->handle();

$dispatcher = $di['dispatcher'];

// Pass the processed router parameters to the dispatcher
$dispatcher->setControllerName($router->getControllerName());
$dispatcher->setActionName($router->getActionName());
$dispatcher->setParams($router->getParams());

// Dispatch the request
$dispatcher->dispatch();

//Get the returned value by the lastest executed action
$response = $dispatcher->getReturnedValue();

//Check if the action returned is a 'response' object
if ($response instanceof Phalcon\Http\ResponseInterface) {

    //Send the request
    $response->send();
}

}

Yet another alternative that catch exceptions produced in the dispatcher forwarding to other actions consequently:
```

```
<?php

// Get the 'router' service
$router = $di['router'];

$router->handle();

$dispatcher = $di['dispatcher'];

// Pass the processed router parameters to the dispatcher
$dispatcher->setControllerName($router->getControllerName());
$dispatcher->setActionName($router->getActionName());
$dispatcher->setParams($router->getParams());

try {

    // Dispatch the request
    $dispatcher->dispatch();

} catch (Exception $e) {

    //An exception has occurred, dispatch some controller/action aimed for that

    // Pass the processed router parameters to the dispatcher
    $dispatcher->setControllerName('errors');
    $dispatcher->setActionName('action503');

    // Dispatch the request
}
```

```
$dispatcher->dispatch();

}

//Get the returned value by the lastest executed action
$response = $dispatcher->getReturnValue();

//Check if the action returned is a 'response' object
if ($response instanceof Phalcon\Http\ResponseInterface) {

    //Send the request
    $response->send();
}
```

Несмотря на то, что этот код более многословен чем код при использовании *Phalcon\ Mvc\ Application*, он предоставляет альтернативу для запуска вашего приложения. В зависимости от своих потребностей, вы, возможно, захотите иметь полный контроль того будет ли создан ответ или нет, или захотите заменить определённые компоненты на свои, либо расширить их функциональность.

## 2.19.4 События приложения

*Phalcon\ Mvc\ Application* может вызывать события *EventsManager* (если они присутствуют). События запускаются с помощью типа “application”. Поддерживаются следующие события:

Название события	Выполняется при
beforeStartModule	До инициализации зарегистрированного модуля
afterStartModule	После инициализации зарегистрированного модуля
beforeHandleRequest	До выполнения цикла диспетчера
afterHandleRequest	После выполнения цикла диспетчера

В примере ниже показано, как указать обработчика событий в компоненте:

```
<?php

use Phalcon\Events\Manager as EventsManager;

$eventsManager = new EventsManager();

$application->setEventManager($eventsManager);

$eventsManager->attach(
    "application",
    function($event, $application) {
        // ...
    }
);
```

## 2.19.5 Внешние источники

- Примеры MVC Github

## 2.20 Маршрутизация (Routing, Роутинг)

Компонент маршрутизации позволяет определять маршруты, которые будут привязаны к контроллерам, или обработчикам для получения запроса. Маршрутизатор просто разбирает URI для определения информации. Маршрутизатор имеет два режима: MVC режим и режим совпадения. Первый режим идеально подходит для работы с MVC приложениями.

### 2.20.1 Определение маршрутов

*Phalcon\ Mvc\ Router* предоставляет расширенные возможности маршрутизации. В MVC режиме вы можете определить маршруты и направить их на контроллеры/действия, которые вам требуются. Маршруты определяются следующим образом:

```
<?php

// Создание маршрутизатора
$router = new \Phalcon\Mvc\Router();

// Определение правила маршрутизации
$router->add(
    "/admin/users/my-profile",
    array(
        "controller" => "users",
        "action"      => "profile",
    )
);

// Еще одно правило
$router->add(
    "/admin/users/change-password",
    array(
        "controller" => "users",
        "action"      => "changePassword",
    )
);

$router->handle();
```

Метод `add()` принимает в качестве первого параметра шаблон ссылки, вторым параметром настройки этого маршрута. В этом случае, если URI соответствует `/admin/users/my-profile`, и будет выполнен контроллер `"users"`, а в нём действие `"profile"`. Маршрутизатор не выполняет действие контроллера, он только собирает эту информацию, чтобы сообщить правильные параметры в компонент *Phalcon\ Mvc\ Dispatcher*.

Приложение может иметь множество маршрутов, определения их по одному может быть достаточно трудоемкой задачей. В таких случаях мы можем создавать более гибкие маршруты:

```
<?php

// Создание маршрутизатора
$router = new \Phalcon\Mvc\Router();

// Определение правила маршрутизации
$router->add(
    "/admin/:controller/a/:action/:params",
    array(

```

```
    "controller" => 1,
    "action"      => 2,
    "params"      => 3,
)
);
```

В примере, приведенном выше, с помощью подстановочных элементов мы делаем маршрут подходящим для множества ссылок. Например, при получении URL (/admin/users/a/delete/dave/301), маршрутизатор разберёт его в:

Контроллер / Controller	users
Действие / Action	delete
Параметр / Parameter	dave
Параметр / Parameter	301

Метод add() принимает шаблон, который по желанию может быть написан с использованием регулярных выражений. Все маршруты должны начинаться с косой черты (/). Регулярные выражения должны соответствовать формату [регулярных выражений PCRE](#). Отметим, что это не нужно добавлять в регулярные выражения разделители. Все маршруты не зависят от регистра.

Второй параметр определяет, какие из подходящих частей следует “привязать” к controller/action/parameters. Соответствующие части берутся из “заполнителей” или по маскам ограничивающимся круглыми скобками. В примере, приведенном выше, первой части соответствует контроллеру (:controller), второй действию и так далее.

Заполнители помогают написанию регулярных выражений, они более читабельны для разработчиков и проще для понимания. Существуют такие заполнители:

Названия контроллеров “camelized”, это означает, что символы (-) и (\_) удаляются, и следующий после них символ преобразуется в верхний регистр. Например, some\_controller преобразуется в SomeController.

Поскольку вы можете использовать множество маршрутов, добавляя их методом add(), порядок, в котором маршруты добавляются указывает их актуальность, последние добавленные маршруты имеют больший приоритет, чем добавленные ранее. Внутри все определенные маршруты перемещаются в обратном порядке, пока [Phalcon Mvc Router](#) не найдёт тот, который соответствует данному URI и использует его, игнорируя остальные.

## Именованные параметры

В примере ниже показано, как определить имена для параметров маршрутов:

```
<?php
$router->add(
    "/news/([0-9]{4})/([0-9]{2})/([0-9]{2})/:params",
    array(
        "controller" => "posts",
        "action"      => "show",
        "year"        => 1, // ([0-9]{4})
        "month"       => 2, // ([0-9]{2})
        "day"         => 3, // ([0-9]{2})
        "params"      => 4, // :params
    )
);
```

В приведенном выше примере, в маршруте не определены части для “контроллера” или “действия”. Эти параметры заменяются фиксированными значениями (“posts” и “show”). Пользователь не будет видеть

вызванный контроллер. Внутри контроллера именованные параметры можно получить следующим образом:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function showAction()
    {

        // Возвращает параметр "year"
        $year = $this->dispatcher->getParam("year");

        // Возвращает параметр "month"
        $month = $this->dispatcher->getParam("month");

        // Возвращает параметр "day"
        $day = $this->dispatcher->getParam("day");
    }
}
```

Обратите внимание, что значения параметров получаются из диспетчера. Это происходит потому, что это компонент, который, непосредственно запускает в работу ваше приложение. Кроме того, существует и другой способ создавать именованные параметры, например, как часть правила маршрутизации:

```
<?php

$router->add(
    "/documentation/{chapter}/{name}.{type:[a-z]+}",
    array(
        "controller" => "documentation",
        "action"      => "show"
    )
);
```

Вы можете получить доступ к их значениям так же, как раньше:

```
<?php

class DocumentationController extends \Phalcon\Mvc\Controller
{

    public function showAction()
    {

        // Возвращает параметр "name"
        $name = $this->dispatcher->getParam("name");

        // Возвращает параметр "type"
        $type = $this->dispatcher->getParam("type");
    }
}
```

```
    }  
}  
}
```

## Краткий синтаксис

Если вам не нравится использование массивов для определения правил маршрута, альтернативный синтаксис также доступен. Следующие примеры дают одинаковый результат:

```
<?php  
  
// Краткий синтаксис  
$router->add("/posts/{year:[0-9]+}/{title:[a-z\-\-]+}", "Posts::show");  
  
// Использование массива  
$router->add(  
    "/posts/([0-9+])/([a-z\-\-]+)",  
    array(  
        "controller" => "posts",  
        "action"      => "show",  
        "year"        => 1,  
        "title"       => 2,  
    )  
);
```

## Совмещение массивов и краткого синтаксиса

Массив и краткий синтаксис может быть смешанным, в данном случае, обратите внимание, что именованные параметры автоматически добавляются в маршрут в соответствии с положением, на котором они были определены:

```
<?php  
  
// В качестве первой позиции выступает параметр 'country'  
$router->add('/news/{country:[a-z]{2}}/([a-z+])/([a-z\-\-]+)',  
    array(  
        'section' => 2, // Это уже позиция номер 2  
        'article'  => 3  
    )  
);
```

## Маршрутизация модулей

Вы можете определить маршруты, пути которых включают в себя модули. Это особенно подходит для мульти-модульных приложений. Возможно так же определить маршрут по умолчанию, который включает в себя модуль шаблона:

```
<?php  
  
$router = new Phalcon\Mvc\Router(false);  
  
$router->add('/:module/:controller/:action/:params', array(  
    'module' => 1,  
    'controller' => 2,
```

```
'action' => 3,
'params' => 4
));
```

В этом случае маршрут всегда должен иметь имя модуля в качестве части URL-адреса. Например, в следующем URL: /admin/users/edit/sonny, будут обработан как:

Модуль	admin
Контроллер	users
Действие	edit
Параметр	sonny

Или вы можете привязать конкретные маршруты к конкретным модулям:

```
<?php

$router->add("/login", array(
    'module' => 'backend',
    'controller' => 'login',
    'action' => 'index',
));
```

```
$router->add("/products/:action", array(
    'module' => 'frontend',
    'controller' => 'products',
    'action' => 1,
));
```

Или привязать к конкретному пространству имен:

```
<?php

$router->add("/:namespace/login", array(
    'namespace' => 1,
    'controller' => 'login',
    'action' => 'index'
));
```

Пространства имён и названия классов должны передаваться раздельно:

```
<?php

$router->add("/login", array(
    'namespace' => 'Backend\Controllers',
    'controller' => 'login',
    'action' => 'index'
));
```

## Разделение по HTTP методам

При добавлении маршрута, используя метод add(), маршрут будет активен для любого HTTP-метода. Иногда можно использовать маршрут для конкретного метода, это особенно полезно при создании RESTful приложений:

```
<?php

// Маршрут сооответствует только HTTP методу GET
$router->addGet("/products/edit/{id}", "Products::edit");
```

```
// Маршрут соответствует только HTTP методу POST  
$router->addPost("/products/save", "Products::save");  
  
// Маршрут соответствует сразу двум HTTP методам POST и PUT  
$router->add("/products/update")->via(array("POST", "PUT"));
```

## Использование преобразований

Метод convert позволяет трансформировать параметры маршрута до передачи их диспетчеру, следующий пример показывает вариант использования:

```
<?php  
  
// Название действия разрешает использование "-": /products/new-ipod-nano-4-generation  
$router  
    ->add('/products/{slug:[a-z\-]+}', array(  
        'controller' => 'products',  
        'action' => 'show'  
    ))  
    ->convert('slug', function($slug) {  
        // Удаляем тире из выбранного параметра  
        return str_replace('-', '', $slug);  
    });
```

## Группы маршрутов

Если наборы маршрутов имеют общие пути, они могут быть сгруппированы для легкой поддержки:

```
<?php  
  
$router = new \Phalcon\Mvc\Router();  
  
// Создаётся группа с общим модулем и контроллером  
$blog = new \Phalcon\Mvc\Router\Group(array(  
    'module' => 'blog',  
    'controller' => 'index'  
));  
  
// Маршруты начинаются с /blog  
$blog->setPrefix('/blog');  
  
// Добавление маршрута в группу  
$blog->add('/save', array(  
    'action' => 'save'  
));  
  
// Еще один маршрут  
$blog->add('/edit/{id}', array(  
    'action' => 'edit'  
));  
  
// Маршрут для действия по умолчанию  
$blog->add('/blog', array(  
    'controller' => 'blog',  
    'action' => 'index'
```

```
));

// Добавление группы в общие правила маршрутизации
$router->mount($blog);
```

Вы можете размещать группы маршрутов в разных файлах приложения, добиваясь оптимальной структуры и чистоты кода:

```
<?php

class BlogRoutes extends Phalcon\Mvc\Router\Group
{
    public function initialize()
    {
        // Параметры по умолчанию
        $this->setPaths(array(
            'module' => 'blog',
            'namespace' => 'Blog\Controllers',
        ));

        // Маршруты начинаются с префикса /blog
        $this->setPrefix('/blog');

        // Добавляем маршрут
        $this->add('/save', array(
            'action' => 'save'
        ));

        // Еще маршрут
        $this->add('/edit/{id}', array(
            'action' => 'edit'
        ));

        // Данные для маршрута по умолчанию
        $this->add('/blog', array(
            'controller' => 'blog',
            'action' => 'index'
        ));
    }
}
```

Созданную группу надо подмонтировать в маршрутизатор:

```
<?php

// Добавляем маршруты в общий маршрутизатор:
$router->mount(new BlogRoutes());
```

## 2.20.2 Соответствие маршрутов

Текущий URI передаётся маршрутизатору для сопоставления его маршруту. По умолчанию, URI для обработки берется из переменной `$_GET['_url']`, полученной с использованием `mod_rewrite`. Для Phalcon подходят очень простые правила `mod_rewrite`:

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-
```

```
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ index.php?_url=/\$1 [QSA,L]
```

В следующем примере показано, как использовать этот компонент автономно:

```
<?php

// Создание маршрутизатора
$route = new \Phalcon\Mvc\Router();

// Тут устанавливаются правила маршрутизации
// ...

// Будет использован $_GET["_url"]
$route->handle();

// Можно указать параметр самостоятельно
$route->handle("/employees/edit/17");

// Получаем выбранный контроллер
echo $route->getControllerName();

// .. и соответствующее действие
echo $route->getActionName();

// Получаем сам выбранный для ссылки маршрут
$route = $route->getMatchedRoute();
```

### 2.20.3 Именованные маршруты

Каждый маршрут, добавленный в маршрутизатор, хранится как объект *Phalcon\ Mvc\ Router\ Route*. Этот класс включает в себя все детали каждого маршрута. Например, мы можем дать ему имя и однозначно идентифицировать в нашем приложении. Это особенно полезно, если вы хотите создать ссылки для него.

```
<?php

$route = $route->add("/posts/{year}/{title}", "Posts::show");

$route->setName("show-posts");

// или проще

$route->add("/posts/{year}/{title}", "Posts::show")->setName("show-posts");
```

Затем, при помощи компонента *Phalcon\ Mvc\ Url* и названия маршрута можно создать ссылку:

```
<?php

// создадим /posts/2012/phalcon-1-0-released
echo $url->get(array(
    "for" => "show-posts",
    "year" => "2012",
    "title" => "phalcon-1-0-released"
));
```

## 2.20.4 Примеры использования

Ниже приведены примеры пользовательских маршрутов:

```
<?php

// пример - "/system/admin/a/edit/7001"
$router->add(
    "/system/:controller/a/:action/:params",
    array(
        "controller" => 1,
        "action"      => 2,
        "params"      => 3
    )
);

// пример - "/es/news"
$router->add(
    "/{[a-z]{2}}/:controller",
    array(
        "controller" => 2,
        "action"      => "index",
        "language"   => 1
    )
);

// пример - "/es/news"
$router->add(
    "/{language:[a-z]{2}}/:controller",
    array(
        "controller" => 2,
        "action"      => "index"
    )
);

// пример - "/admin/posts/edit/100"
$router->add(
    "/admin/:controller/:action/:int",
    array(
        "controller" => 1,
        "action"      => 2,
        "id"         => 3
    )
);

// пример - "/posts/2010/02/some-cool-content"
$router->add(
    "/posts/{[0-9]{4}}/({[0-9]{2}})/({[a-z\-.]+})",
    array(
        "controller" => "posts",
        "action"      => "show",
        "year"        => 1,
        "month"       => 2,
        "title"       => 4
    )
);

// пример - "/manual/en/translate.adapter.html"
$router->add(
```

```
"/manual/([a-z]{2})/([a-z\.]+)\.html",
array(
    "controller" => "manual",
    "action"      => "show",
    "language"    => 1,
    "file"        => 2
)
);

// пример - /feed/fr/le-robots-hot-news.atom
$router->add(
    "/feed/{lang:[a-z]+}/{blog:[a-z\.-]+}\.{type:[a-z\.-]+}",
    "Feed::get"
);

// пример - /api/v1/users/peter.json
$router->add('/api/(v1|v2)/{method:[a-z]+}/{param:[a-z]+}\.(json|xml)',
array(
    'controller' => 'api',
    'version'     => 1,
    'format'      => 4
)
);

```

Остерегайтесь использования спецсимволов в регулярных выражениях для контроллеров и пространств имён. Эти параметры формируют имена классов и файлов, что, в свою очередь, взаимодействует с файловой системой, и может использоваться злоумышленником для чтения несанкционированных файлов. Безопасным является регулярное выражение: /([a-zA-Z0-9\_-]+)

## 2.20.5 Поведение по умолчанию

У компонента *Phalcon\ Mvc\ Router* есть поведение по умолчанию, при котором все URL обрабатываются по простому шаблону: `:/controller/:action/:params`

Например, ссылку вида `http://phalconphp.com/documentation/show/about.html` маршрутизатор проанализирует как:

Контроллер	documentation
Действие	show
Параметр	about.html

Если вы не хотите использовать маршруты по умолчанию в вашем приложении, вы должны указать `false` в качестве параметра при создании объекта маршрутизатора:

```
<?php

// Создания маршрутизатора без поддержки стандартной маршрутизации
$router = new \Phalcon\ Mvc\ Router(false);
```

## 2.20.6 Указание маршрута по умолчанию

При обращению к главной странице приложения срабатывает маршрут `'/'`, в нём надо указать что должно срабатывать:

```
<?php

$router->add("/", array(
    'controller' => 'index',
    'action' => 'index'
));
```

## 2.20.7 404 страница

Если ни один из указанных маршрутов в маршрутизаторе не совпадёт, вы можете определить действие для этого случая:

```
<?php

// Указание действия для 404 страницы
$router->notFound(array(
    "controller" => "index",
    "action" => "route404"
));
```

## 2.20.8 Установка параметров по умолчанию

Можно определить значения по умолчанию для некоторых частей маршрута, таких как модуль, контроллер или действие. Когда в маршруте отсутствует любая из указанных частей, они будут автоматически заполнены маршрутизатором из значений по умолчанию:

```
<?php

// Установка по умолчанию
$router->setDefaultModule('backend');
$router->setDefaultNamespace('Backend\Controllers');
$router->setDefaultController('index');
$router->setDefaultAction('index');

// Используя значения массива
$router->setDefaults(array(
    'controller' => 'index',
    'action' => 'index'
));
```

## 2.20.9 Использование конечного /

Иногда обращение к маршруту может быть с дополнительной косой чертой (слэш) и в конце маршрута, это в отдельных случаях может привести к несоответствию маршруту. Вы можете настроить маршрутизатор для автоматического удаления слэша из конца обрабатываемого маршрута:

```
<?php

$router = new \Phalcon\Mvc\Router();

// Конечные косые черты будут автоматически удалены
$router->removeExtraSlashes(true);
```

Или вы можете изменить определенные маршруты, в которых необходимо использовать косые черты:

```
<?php

$router->add(
    '/{language:[a-z]{2}}/{controller[/]{0,1}}',
    array(
        'controller' => 2,
        'action'      => 'index'
    )
);
```

## 2.20.10 Дополнительные условия

Иногда требуется, чтобы перед выполнением маршрута удовлетворял определённым условиям. Вы можете добавлять произвольные условия используя функцию обратного вызова (callback) ‘beforeMatch’. Если эта функция вернёт false, то запрос не совпадёт с условием и маршрут не выполнится:

```
<?php

$router->add('/login', array(
    'module' => 'admin',
    'controller' => 'session'
))->beforeMatch(function($uri, $route) {
    // Проверим, что это был Ajax-запрос
    if ($_SERVER['X_REQUESTED_WITH'] == 'xmlhttprequest') {
        return false;
    }
    return true;
});
```

Вы можете повторно использовать эти дополнительные условия в классах:

```
<?php

class AjaxFilter
{
    public function check()
    {
        return $_SERVER['X_REQUESTED_WITH'] == 'xmlhttprequest';
    }
}
```

И использовать этот класс вместо анонимной функции:

```
<?php

$router->add('/get/info/{id}', array(
    'controller' => 'products',
    'action'      => 'info'
))->beforeMatch(array(new AjaxFilter(), 'check'));
```

## 2.20.11 Ограничение по имени хоста

Маршрутизатор позволяет вам выставлять ограничения по имени хоста. Это означает, что конкретные маршруты или группы маршрутов могут быть привязаны к конкретным именам хостов:

```
<?php

$router->add('/login', array(
    'module' => 'admin',
    'controller' => 'session',
    'action' => 'login'
))->setHostName('admin.company.com');
```

Имя хоста так же может быть регулярным выражением:

```
<?php

$router->add('/login', array(
    'module' => 'admin',
    'controller' => 'session',
    'action' => 'login'
))->setHostName('([a-z+]).company.com');
```

В группах маршрутов вы можете установить ограничение по имени хоста, которое будет применяться к каждому маршруту в группе:

```
<?php

// Создаём группу с общим модулем и контроллером
$blog = new \Phalcon\Mvc\Router\Group(array(
    'module' => 'blog',
    'controller' => 'posts'
));

// Ограничиваем по имени хоста
$blog->setHostName('blog.mycompany.com');

// Все маршруты начинаются с /blog
$blog->setPrefix('/blog');

// Маршрут по умолчанию
$blog->add('/', array(
    'action' => 'index'
));

// Добавляем маршрут в группу
$blog->add('/save', array(
    'action' => 'save'
));

// Добавляем ещё один маршрут в группу
$blog->add('/edit/{id}', array(
    'action' => 'edit'
));

// Добавляем группу в маршрутизатор
$router->mount($blog);
```

## 2.20.12 Источники URI

По умолчанию текущий URI для обработки берётся из переменной `$_GET['_url']`, так устроено внутри Phalcon и стандартных правилах mod-rewrite, очень просто можно указать использование для этих

целей переменную `$_SERVER['REQUEST_URI']`:

```
<?php
```

```
$router->setUriSource(Router::URI_SOURCE_GET_URL); // использование $_GET['_url'] (по умолчанию)  
$router->setUriSource(Router::URI_SOURCE_SERVER_REQUEST_URI); // использование $_SERVER['REQUEST_URI'] (по умолчанию)
```

Или вы можете самостоятельно передавать URI в метод “handle”:

```
<?php
```

```
$router->handle('/some/route/to/handle');
```

## 2.20.13 Тестирование маршрутов

Компонент маршрутизации не имеет внутренних зависимостей, вы можете создать файл, как показано ниже, для проверки своих маршрутов:

```
<?php
```

```
// Маршруты для проверки  
$testRoutes = array(  
    '/',  
    '/index',  
    '/index/index',  
    '/index/test',  
    '/products',  
    '/products/index/',  
    '/products/show/101',  
);  
  
$router = new Phalcon\Mvc\Router();  
  
// Тут необходимо установить правила маршрутизации  
//...  
  
// Цикл проверки маршрутов  
foreach ($testRoutes as $testRoute) {  
  
    // Обработка маршрута  
    $router->handle($testRoute);  
  
    echo 'Тестирование ', $testRoute, '<br>;  
  
    // Проверка выбранного маршрута  
    if ($router->wasMatched()) {  
        echo 'Контроллер (Controller): ', $router->getControllerName(), '<br>;  
        echo 'Действие (Action): ', $router->getActionName(), '<br>;  
    } else {  
        echo 'Маршрут не поддерживается<br>;  
    }  
    echo '<br>;  
}
```

## 2.20.14 Маршруты на аннотациях

Компонент `Phalcon\ Mvc\ Router\ Annotations` интегрирован с компонентом `annotations`, и позволяет получать информацию о маршрутах из doc-блоков внутри кода контроллеров. Используя эту стратегию, вы можете указывать маршруты непосредственно в контроллерах, вместо того, чтобы указывать их в отдельных правилах маршрутизации:

```
<?php

$di['router'] = function() {

    // Используем маршрутизатор на аннотациях
    $router = new \Phalcon\ Mvc\ Router\ Annotations(false);

    // Чтение аннотаций из контроллера ProductsController для ссылок начинающихся на /api/products
    $router->addResource('Products', '/api/products');

    return $router;
};
```

Аннотации могут быть определены следующим образом:

```
<?php

/**
 * @RoutePrefix("/api/products")
 */
class ProductsController
{

    /**
     * @Get("/")
     */
    public function indexAction()
    {

    }

    /**
     * @Get("/edit/{id:[0-9]+}", name="edit-robot")
     */
    public function editAction($id)
    {

    }

    /**
     * @Route("/save", methods={"POST", "PUT"}, name="save-robot")
     */
    public function saveAction()
    {

    }

    /**
     * @Route("/delete/{id:[0-9]+}", methods="DELETE",
     *        conversors={"id":MyConversors::checkId"})
     */
    public function deleteAction($id)
```

```
{
}

public function infoAction($id)
{
}

}
```

Маршрутизатор поддерживает только строго определённые методы, вот список текущих поддерживающих аннотаций:

Название	Описание	Использование
RoutePrefix	Предфикс добавляемый к каждому маршруту. Эта аннотация должна быть в комментариях класса (контроллера)	@RoutePrefix("/api/products")
Route	Эта аннотация создаёт маршрут для метода, она должна быть в комментариях метода.	@Route("/api/products/show")
Get	Эта аннотация создаёт маршрут для метода, разрешается только HTTP метод GET	@Get("/api/products/search")
Post	Эта аннотация создаёт маршрут для метода, разрешается только HTTP метод POST	@Post("/api/products/save")
Put	Эта аннотация создаёт маршрут для метода, разрешается только HTTP метод PUT	@Put("/api/products/save")
Delete	Эта аннотация создаёт маршрут для метода, разрешается только HTTP метод DELETE	@Delete("/api/products/delete/{id}")
Options	Эта аннотация создаёт маршрут для метода, разрешается только HTTP метод OPTIONS	@Option("/api/products/info")

Для аннотации при добавлении маршрутов поддерживаются следующие параметры:

Название	Описание	Использование
methods	Определяет HTTP метод доступа к маршруту	@Route("/api/products", methods={"GET", "POST"})
name	Определяет название маршрута	@Route("/api/products", name="get-products")
paths	Массив дополнительных частей пути Phalcon\Mvc\Router::add	@Route("/posts/{id}/{slug}", paths={"module": "backend"})
conversors	Метод преобразования для применения к параметрам	@Route("/posts/{id}/{slug}", conversors={"id": "MyConversor::getId"})

Для формирования маршрутов из контроллеров модулей стоит использовать метод addModuleResource:

```
<?php
```

```
$di['router'] = function() {

    // Используем маршрутизатор на аннотациях
    $router = new \Phalcon\Mvc\Router\Annotations(false);

    // Чтение аннотаций из контроллера Backend\Controllers\ProductsController для ссылок начинающихся на /api/products
    $router->addModuleResource('backend', 'Products', '/api/products');
```

```

    return $router;
};

```

### 2.20.15 Создание собственного маршрутизатора

Для создания адаптера необходимо реализовать интерфейс *Phalcon\ Mvc\ RouterInterface*. Созданным классом надо подменить маршрутизатор ('router') в момент инициализации приложения.

## 2.21 Диспетчер контроллеров

Компонент *Phalcon\ Mvc\ Dispatcher* отвечает за инициализацию контроллеров и выполнения в них действий, для MVC приложения. Понимание его работы и его возможностей помогает нам получить больше возможностей предоставляемых фреймворком.

### 2.21.1 Цикл работы диспетчера

Это важнейший процесс, который имеет много общего с работой MVC, особенно в части работы контроллеров. Работа контроллера вызывается диспетчером. Файлы контроллерачитываются, загружаются, инициализируются, чтобы затем выполнить необходимые действия. Если действие направляет поток на другой контроллер/действие (action), диспетчер контроллера стартует снова. Для лучшей иллюстрации в примере ниже показан приблизительный процесс происходящий внутри *Phalcon\ Mvc\ Dispatcher*:

```

<?php

// Цикл диспетчера
while (!$finished) {

    $finished = true;

    $controllerClass = $controllerName . "Controller";

    // Создание экземпляра класса контроллера, с помощью автозагрузчика
    $controller = new $controllerClass();

    // Выполнение действия
    call_user_func_array(array($controller, $actionName . "Action"), $params);

    // Значение переменной должно быть изменено при необходимости запуска другого контроллера
    $finished = true;

}

```

Этот код, безусловно, нуждается в дополнительных проверках и доработке, но здесь наглядно показана типичная последовательность операций в диспетчере.

### События при работе диспетчера

*Phalcon\ Mvc\ Dispatcher* может отправлять события *EventsManager* если это необходимо. События вызываются с помощью типа "dispatch". Некоторые события, при возвращении false, могут остановить активную операцию. Поддерживаются следующие события:

Название события	Время срабатывания	Прерывает операцию?	Triggered to
beforeDispatch	После старта цикла диспетчера. В этот момент диспетчер не знает, существуют ли контроллеры или действия, которые должны быть выполнены. Диспетчер владеет только информацией поступившей из маршрутизатора	Да	Listeners
beforeDispatchTo	После выполнения цикла диспетчера. В этот момент диспетчер не знает, существуют ли контроллеры или действия, которые должны быть выполнены. Диспетчер знает только информацию, поступившую из маршрутизатора	Да	Listeners
beforeExecuteRoute	После выполнения действия в контроллере. В этой точке контроллер инициализирован и знает о существовании действия (action)	Да	Listeners/Controllers
afterExecuteRoute	После выполнения действия в контроллере. Не останавливает текущую операцию, используйте это событие только для завершения/очистки после выполненного действия	Нет	Controllers
beforeNotFoundAction	Когда действие не найдено в контроллере	Нет	Listeners/Controllers
beforeException	После вызова диспетчером любого исключения	Да	Listeners
afterDispatch	После выполнения цикла диспетчера. Не останавливает текущую операцию, используйте это событие только для завершения/очистки после выполненного действия	Да	Listeners
afterDispatchOnException	После завершения цикла диспетчера	Нет	Listeners

В обучающем материале [INVO](#) показано, как воспользоваться диспетчером событий для реализации фильтра безопасности [Acl](#).

В примере ниже показано как прикрепить слушателей (listeners) к событиям контроллера:

```
<?php

use Phalcon\Mvc\Dispatcher as MvcDispatcher,
    Phalcon\Events\Manager as EventsManager;

$di->set('dispatcher', function(){

    // Создание менеджера событий
    $eventsManager = new EventsManager();

    // Прикрепление функции-слушателя для событий типа "dispatch"
    $eventsManager->attach("dispatch", function($event, $dispatcher) {
        //...
    });

    $dispatcher = new MvcDispatcher();

    // Связывание менеджера событий с диспетчером
    $dispatcher->setEventsManager($eventsManager);

    return $dispatcher;
}, true);
```

Экземпляр контроллера автоматически выступает в качестве слушателя для событий, так что вы можете реализовать методы в самом контроллере:

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function beforeExecuteRoute($dispatcher)
    {
        // Выполняется перед каждым найденным действием
    }

    public function afterExecuteRoute($dispatcher)
    {
        // Выполняется после каждого выполненного действия
    }

}
```

## 2.21.2 Переадресация на другое действие

Цикл диспетчера позволяет перенаправить поток на другой контроллер/действие. Это очень полезно, для проверки может ли пользователь иметь доступ к определенным функциям, перенаправления пользователя на другую страницу или просто для повторного использования кода.

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function saveAction($year, $postTitle)
    {

        // .. сохраняем данные и перенаправляем пользователя

        // Перенаправляем на действие index
        $this->dispatcher->forward(array(
            "controller" => "post",
            "action" => "index"
        ));
    }
}
```

Имейте ввиду, использование метода “forward” - это не то же самое что редирект в HTTP. Хотя внешне результат будет таким же. Метод “forward” не перезагружает текущую страницу, все перенаправления выполняются в одном запросе, тогда как HTTP редирект требует два запроса для завершения процесса.

Пример перенаправлений:

```
<?php

// Направляем поток на другое действие текущего контроллера
$this->dispatcher->forward(array(
```

```
"action" => "search"
));
// Направляем поток на другое действие текущего контроллера с передачей параметров
$this->dispatcher->forward(array(
    "action" => "search",
    "params" => array(1, 2, 3)
));
```

Метод `forward` принимает следующие параметры:

Параметр	Описание
controller	Правильное имя контроллера для вызова
action	Правильное название действия для вызова
params	Массив параметров для действия (action)
namespace	Пространство имён, которому принадлежит контроллер

### 2.21.3 Получение параметров

Если текущий маршрут содержит именованные параметры, вы можете получить их в контроллере, представлении или любом другом компоненте, расширяющим [Phalcon|DI|Injectable](#).

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function saveAction()
    {

        // Получение параметра title, находящимся в параметрах URL
        $title = $this->dispatcher->getParam("title");

        // Получение параметра year, пришедшего из URL и отфильтрованного как число
        $year = $this->dispatcher->getParam("year", "int");
    }
}
```

### 2.21.4 Обработка исключений “Не найдено”

Используйте возможности [EventsManager](#) для установки событий, выполняемых при отсутствии требуемого контроллера/действия.

```
<?php

use Phalcon\Dispatcher,
    Phalcon\Mvc\Dispatcher as MvcDispatcher,
    Phalcon\Events\Manager as EventsManager;

$di->set('dispatcher', function() {
```

```
// Создаем менеджер событий
$eventsManager = new EventsManager();

// Прикрепляем слушателя
$eventsManager->attach("dispatch:beforeException", function($event, $dispatcher, $exception) {

    switch ($exception->getCode()) {
        case Dispatcher::EXCEPTION_HANDLER_NOT_FOUND:
        case Dispatcher::EXCEPTION_ACTION_NOT_FOUND:
            $dispatcher->forward(array(
                'controller' => 'index',
                'action' => 'show404'
            ));
            return false;
    }
});

$dispatcher = new MvcDispatcher();

//Прикрепляем менеджер событий к диспетчеру
$dispatcher->setEventsManager($eventsManager);

return $dispatcher;

}, true);

```

## 2.21.5 Реализация собственных диспетчеров

Для создания диспетчеров необходимо реализовать интерфейс `Phalcon\ Mvc\ DispatcherInterface` и подменить диспетчер Phalcon.

# 2.22 Микроприложения

С помощью Phalcon можно создавать приложения по типу “Микрофреймворк”. Для этого, необходимо написать всего лишь несколько строк кода. Микроприложения подходят для реализации небольших приложений, различных API и прототипов на практике.

```
<?php

$app = new Phalcon\ Mvc\ Micro();

$app->get('/say/welcome/{name}', function ($name) {
    echo "<h1>Welcome $name!</h1>";
});

$app->handle();

```

## 2.22.1 Создание микроприложения

`Phalcon\ Mvc\ Micro` это класс, отвечающий за реализацию микроприложения.

```
<?php  
  
$app = new Phalcon\Mvc\Micro();
```

## 2.22.2 Создание путей

После создания экземпляра класса необходимо добавить некоторые пути. *Phalcon\ Mvc\ Router* отвечает за управление путями, которые должны всегда начинаться с /. При создании путей необходимо указывать, какой метод HTTP используется, чтобы запросы путей соответствовали методам HTTP. Ниже представлен пример, показывающий как создавать пути, используя метод GET:

```
<?php  
  
$app->get('/say/hello/{name}', function ($name) {  
    echo "<h1>Hello! $name</h1>";  
});
```

Метод “get” показывает, что используется GET-запрос. Путь /say/hello/{name} также имеет параметр {\$name}, который напрямую передается обработчику пути (анонимная функция). Обработка пути выполняется, когда путь совпадает. Обработчик может быть любого типа, который возвращает данные в PHP-среде. Следующий пример демонстрирует, как создавать различные типы обработчиков пути:

```
<?php  
  
// С помощью функции  
function say_hello($name) {  
    echo "<h1>Hello! $name</h1>";  
}  
  
$app->get('/say/hello/{name}', "say_hello");  
  
// С помощью статического метода  
$app->get('/say/hello/{name}', "SomeClass::someSayMethod");  
  
// С помощью метода объекта  
$myController = new MyController();  
$app->get('/say/hello/{name}', array($myController, "someAction"));  
  
// Анонимная функция (замыкание)  
$app->get('/say/hello/{name}', function ($name) {  
    echo "<h1>Hello! $name</h1>";  
});
```

*Phalcon\ Mvc\ Micro* предлагает набор инструментов для создания HTTP-метода (или методов), необходимых для создания пути:

```
<?php  
  
// Соедиаем, если HTTP-метод - GET  
$app->get('/api/products', "get_products");  
  
// Соедиаем, если HTTP-метод - POST  
$app->post('/api/products/add', "add_product");  
  
// Соедиаем, если HTTP-метод - PUT  
$app->put('/api/products/update/{id}', "update_product");
```

```
// Соединим, если HTTP-метод - DELETE
$app->delete('/api/products/remove/{id}', "delete_product");

// Соединим, если HTTP-метод - OPTIONS
$app->options('/api/products/info/{id}', "info_product");

// Соединим, если HTTP-метод - PATCH
$app->patch('/api/products/update/{id}', "info_product");

// Соединим, если HTTP-метод - GET или POST
$app->map('/repos/store/refs', "action_product")->via(array('GET', 'POST'));
```

## Пути с параметрами

Создание параметров путей - довольно простая задача, как показывает пример выше. Имя параметра должно находиться в скобках. Параметры также можно задавать с помощью регулярных выражений для того, чтобы быть уверенными в наличии данных. Это показано в примере ниже:

```
<?php

// Данный путь имеет два параметра, у каждого из которых задан формат
$app->get('/posts/{year:[0-9]+}/{title:[a-zA-Z\-\-]+}', function ($year, $title) {
    echo "<h1>Title: $title</h1>";
    echo "<h2>Year: $year</h2>";
});
```

## Маршрут по умолчанию

Как правило, маршрутом по умолчанию в приложении является маршрут /. Чаще всего, обращения будут идти именно к нему через метод GET. Этот сценарий можно описать следующим образом:

```
<?php

// Это маршрут по умолчанию
$app->get('/', function () {
    echo "<h1>Welcome!</h1>";
});
```

## Правила перезаписи (Rewrite Rules)

Следующие правила могут быть использованы вместе с Apache для перезаписи URI:

```
<IfModule mod_rewrite.c>
    RewriteEngine On
    RewriteCond %{REQUEST_FILENAME} !-f
    RewriteRule ^(.*)$ index.php?url=/{$1} [QSA,L]
</IfModule>
```

### 2.22.3 Работа с заголовками ответов (Responses)

Вы можете работать с любыми заголовками ответов в обработчике: сразу сделать вывод, использовать шаблонизатор, подключить шаблонизатор, вернуть JSON и т.д.:

```
<?php

// Прямой вывод
$app->get('/say/hello', function () {
    echo "<h1>Hello! $name</h1>";
});

// Подключение внешнего файла
$app->get('/show/results', function () {
    require 'views/results.php';
});

// Возврат JSON
$app->get('/get/some-json', function () {
    echo json_encode(array("some", "important", "data"));
});
```

В дополнение к этому, у вас есть доступ к сервису “*response*”, благодаря которому вы можете обрабатывать ответы ещё более гибко:

```
<?php

$app->get('/show/data', function () use ($app) {

    // Установка заголовка Content-Type
    $app->response->setContentType('text/plain')->sendHeaders();

    // Вывод содержимого файла
    readfile("data.txt");

});
```

Или создайте объект класса Response и верните его из обработчика:

```
<?php

$app->get('/show/data', function () {

    // Создаем объект для работы с заголовками ответов
    $response = new Phalcon\Http\Response();

    // Установка заголовка Content-Type
    $response->setContentType('text/plain');

    // Передаем содержимое файла
    $response->setContent(file_get_contents("data.txt"));

    // Возвращаем объект Response
    return $response;
});
```

## 2.22.4 Создание перенаправлений (Redirects)

Перенаправления могут быть использованы для того, чтобы перенаправить поток исполнения на другой маршрут:

```
<?php

// Этот маршрут выполняет перенаправление на другой маршрут
$app->post('/old/welcome', function () use ($app) {
    $app->response->redirect("new/welcome")->sendHeaders();
});

$app->post('/new/welcome', function () use ($app) {
    echo 'This is the new Welcome';
});
```

## 2.22.5 Создание URL-адресов для маршрутов

Класс *Phalcon\ Mvc\Url* может быть использован для получения URL-адреса на основе определенных маршрутов. Вам нужно создать имя для маршрута; опираясь на него служба “url” выполнить соответствующий URL:

```
<?php

// Установка маршрута с именем "show-post"
$app->get('/blog/{year}/{title}', function ($year, $title) use ($app) {
    //.. здесь показываем текст статьи
})->setName('show-post');

// Где-нибудь используем наш новый адрес
$app->get('/', function() use ($app) {

    echo '<a href="', $app->url->get(array(
        'for' => 'show-post',
        'title' => 'php-is-a-great-framework',
        'year' => 2012
    )), '">Show the post</a>';
});
```

## 2.22.6 Работа с Внедрением зависимостей (Dependency Injector)

В микроприложении сервисы контейнера *Phalcon\DI\FactoryDefault* создаются неявно; Кроме того, вы можете создать за пределами своего приложения контейнер, который будет манипулировать этими сервисами:

```
<?php

use Phalcon\DI\FactoryDefault,
    Phalcon\Mvc\Micro,
    Phalcon\Config\Adapter\Ini as IniConfig;

$di = new FactoryDefault();

$di->set('config', function() {
    return new IniConfig("config.ini");
});
```

```
$app = new Micro();

$app->setDI($di);

$app->get('/', function () use ($app) {
    // Читаем свойства нашего конфигурационного файла
    echo $app->config->app_name;
});

$app->post('/contact', function () use ($app) {
    $app->flash->success('Yes!, the contact was made!');
});
```

Синтаксис массивов удобен для установки/получения сервисов из внутреннего контейнера сервисов:

```
<?php

use Phalcon\Mvc\Micro,
    Phalcon\Db\Adapter\Pdo\Mysql as MysqlAdapter;

$app = new Micro();

// Установка сервиса базы данных
$app['db'] = function() {
    return new MysqlAdapter(array(
        "host" => "localhost",
        "username" => "root",
        "password" => "secret",
        "dbname" => "test_db"
    ));
};

$app->get('/blog', function () use ($app) {
    $news = $app['db']->query('SELECT * FROM news');
    foreach ($news as $new) {
        echo $new->title;
    }
});
```

## 2.22.7 Обработка исключений “Не найдено”

Когда пользователь пытается получить доступ к маршруту, который не определён, микроприложение запускает обработчик “Не найдено”. Пример:

```
<?php

$app->notFound(function () use ($app) {
    $app->response->setStatusCode(404, "Not Found")->sendHeaders();
    echo 'This is crazy, but this page was not found!';
});
```

## 2.22.8 Модели в микроприложениях

*Модели* в микроприложениях работают так же, как и в обычных. Главное - зарегистрировать автозагрузчик:

```
<?php

$loader = new \Phalcon\Loader();

$loader->registerDirs(array(
    __DIR__ . '/models/',
))->register();

$app = new \Phalcon\Mvc\Micro();

$app->get('/products/find', function(){
    foreach (Products::find() as $product) {
        echo $product->name, '<br>';
    }
});

$app->handle();
```

## 2.22.9 События микроприложения

*Phalcon\ Mvc\ Micro* может посыпать события в *EventsManager* (если он присутствует). Events are triggered using the type “micro”. The following events are supported:

Имя события	Действие	Можно ли оставить операцию?
beforeHandle	Решивший метод вызван, в этот момент приложение не знает, есть ли соответствующий маршрут	Да
beforeExecute	Коответствующий маршрут найден и содержит верный обработчик, в этот момент обработчик не будет выполнен	Да
afterExecute	Запускается после запуска обработчика	Нет
beforeNotFound	Запускается, когда каждый из определённых маршрутов удовлетворяет URI	Да
afterHandleResult	Запускается после успешного выполнения всего процесса	Да

В приведённом примере объясняется, как управлять безопасностью приложения используя события:

```
<?php

use Phalcon\Mvc\Micro,
    Phalcon\Events\Manager as EventsManager;

// Создаём менеджер событий
$eventManager = new EventsManager();

// Слушаем все события приложения
$eventManager->attach('micro', function($event, $app) {

    if ($event->getType() == 'beforeExecuteRoute') {
        if ($app->session->get('auth') == false) {

            $app->flashSession->error("The user isn't authenticated");
            $app->response->redirect("/")->sendHeaders();

            // Возвращаем (false) останов операции
        }
    }
});
```

```
        return false;
    }
}

};

$app = new Micro();

// Привязываем менеджер событий к приложению
$app->setEventsManager($eventManager);
```

## 2.22.10 Промежуточные события

В дополнение к менеджеру событий, события могут быть добавлены с использованием методов ‘before’, ‘after’ и ‘finish’:

```
<?php

$app = new Phalcon\Mvc\Micro();

// Выполнится до того, как выполнится любой из маршрутов
// Возврат false отменит выполнение маршрута
$app->before(function() use ($app) {
    if ($app['session']->get('auth') == false) {
        return false;
    }
    return true;
});

$app->map('/api/robots', function(){
    return array(
        'status' => 'OK'
    );
});

$app->after(function() use ($app) {
    // Это выполнится после того, как выполнится маршрут
    echo json_encode($app->getReturnedValue());
});

$app->finish(function() use ($app) {
    // Это выполнится после того, как был обработан запрос
});
```

Вы можете вызывать методы несколько раз, чтобы добавлять больше событий того же типа:

```
<?php

$app->finish(function() use ($app) {
    //First 'finish' middleware
});

$app->finish(function() use ($app) {
    //Second 'finish' middleware
});
```

Код из связанных событий может быть повторно использован в отдельных классах:

```
<?php

use Phalcon\Mvc\Micro\MiddlewareInterface;

/**
 * CacheMiddleware
 *
 * Кэширует страницы для ускорения работы
 */
class CacheMiddleware implements MiddlewareInterface
{
    public function call($application)
    {

        $cache = $application['cache'];
        $router = $application['router'];

        $key = preg_replace('/~[a-zA-Z0-9]/', '', $router->getRewriteUri());

        // Проверяем, закэширован ли запрос
        if ($cache->exists($key)) {
            echo $cache->get($key);
            return false;
        }

        return true;
    }
}
```

Далее передаём экземпляр объекта в приложение:

```
<?php

$app->before(new CacheMiddleware());
```

Доступные следующие промежуточные события:

Имя события	Действие	Можно ли оставить операцию?
before	Перед вызовом обработчика. Может быть использован для управления доступом к приложению	Да
after	Выполняется после вызова обработчика. Может быть использован для подготовки ответа	Нет
finish	Выполняется после отправки ответа. Может быть использован для очистки	Нет

## 2.22.11 Использование контроллеров и обработчиков

При создании приложений среднего уровня через Micro\mvc может потребоваться определённой организации обработчиков в контроллерах. Вы можете использовать *Phalcon\mvc\micro\collection*, чтобы группировать обработчики в контроллерах:

```
<?php

use Phalcon\Mvc\Micro\Collection as MicroCollection;

$post = new MicroCollection();
```

```
// Устанавливаем главный обработчик, например, экземпляр объекта контроллера
$posts->setHandler(new PostsController());

// Устанавливаем общий префикс для всех маршрутов
$posts->setPrefix('/posts');

// Используем метод 'index' в контроллере PostsController
$posts->get('/', 'index');

// Используем метод 'show' в контроллере PostsController
$posts->get('/show/{slug}', 'show');

$app->mount($posts);
```

Контроллер ‘PostsController’ может выглядеть так:

```
<?php

class PostsController extends Phalcon\Mvc\Controller
{

    public function index()
    {
        //...
    }

    public function show($slug)
    {
        //...
    }
}
```

Экземпляр драйвера инициализирован, Коллекция так же может загружать драйверы, если совпал маршрут:

```
<?php

$posts->setHandler('PostsController', true);
$posts->setHandler('Blog\Controllers\PostsController', true);
```

## 2.22.12 Возврат заголовков ответов (Responses)

Обработчики могут возвращать ответы при помощи *Phalcon\Http\Response* или компонента, который реализует соответствующий интерфейс:

```
<?php

use Phalcon\Mvc\Micro,
    Phalcon\Http\Response;

$app = new Micro();

// Возвращаем ответ
$app->get('/welcome/index', function() {

    $response = new Response();
```

```

$response->setStatusCode(401, "Unauthorized");

$response->setContent("Access is not authorized");

return $response;
});

```

### 2.22.13 Отрисовка представлений

Класс `Phalcon\Mvc\View\Simple` может быть использован для отрисовки представлений. Следующий пример показывает как именно:

```

<?php

$app = new Phalcon\Mvc\Micro();

$app['view'] = function() {
    $view = new \Phalcon\Mvc\View();
    $view->setViewsDir('app/views/');
    return $view;
};

// Возвращаем отрисованное представление
$app->get('/products/show', function() use ($app) {

    // Отрисовываем представление app/views/products/show.phtml с передачей в него некоторых переменных
    echo $app['view']->render('products/show', array(
        'id' => 100,
        'name' => 'Artichoke'
    ));
});


```

### 2.22.14 Внешние источники

- Создание простейшего *REST API* урок, показывающий как создать микроприложение, представляющее RESTful API.
- Магазин наклеек очень просто микроприложение [[Github](#)].

## 2.23 Работа с пространством имён

Пространства имён могут быть использованы для исключения пересечений названий классов; это означает, что если в вашем приложении два контроллера с одинаковыми именами, пространства имен могут использоваться, чтобы различать их. Пространства имен также полезны для создания бандлов (bundles) или модулей.

### 2.23.1 Настройка фреймворка

Использование пространств имен накладывает некоторые последствия на загрузку соответствующего контроллера. Для настройки работы фреймворка с пространством имен необходимо выполнить одно или все из следующих задач:

Использовать автозагрузку с учетом пространства имен, например как в Phalcon\Loader:

```
<?php

$loader->registerNamespaces(
    array(
        'Store\Admin\Controllers' => '../bundles/admin/controllers/',
        'Store\Admin\Models'      => '../bundles/admin/models/',
    )
);
```

Использовать в роутинге, как отдельный параметр маршрутизации пути:

```
<?php

$router->add(
    '/admin/users/my-profile',
    array(
        'namespace'  => 'Store\Admin',
        'controller' => 'Users',
        'action'     => 'profile',
    )
);
```

Использовать как часть маршрута:

```
<?php

$router->add(
    '/:namespace/admin/users/my-profile',
    array(
        'namespace'  => 1,
        'controller' => 'Users',
        'action'     => 'profile',
    )
);
```

Если в вашем приложении используется единое пространство имён для контроллеров, то вы можете определить пространство имен по умолчанию в диспетчере. Делая это, вам не потребуется указывать полное имя класса в пути маршрутизатора:

```
<?php

// Регистрация диспетчера
$di->set('dispatcher', function() {
    $dispatcher = new \Phalcon\Mvc\Dispatcher();
    $dispatcher->setDefaultNamespace('Store\Admin\Controllers');
    return $dispatcher;
});
```

## 2.23.2 Контроллеры в пространстве имён

В следующем примере показано, как использовать контроллер, который использует пространство имен:

```
<?php

namespace Store\Admin\Controllers;
```

```
class UsersController extends \Phalcon\Mvc\Controller
{
    public function indexAction()
    {
    }

    public function profileAction()
    {
    }
}
```

### 2.23.3 Модели в пространстве имён

Примите во внимание при использовании модели в пространстве имен следующее:

```
<?php

namespace Store\Models;

class Robots extends Phalcon\Mvc\Model
{
```

Если модели имеют связи с другими моделями, то они тоже должны быть включены в пространство имен:

```
<?php

namespace Store\Models;

class Robots extends Phalcon\Mvc\Model
{
    public function initialize()
    {
        $this->hasMany('id', 'Store\Models\Parts', 'robots_id', array(
            'alias' => 'parts'
        ));
    }
}
```

В PHQL вы должны писать запросы с указанием пространства имен:

```
<?php

$phql = 'SELECT r.* FROM Store\Models\Robots r JOIN Store\Models\Parts p';
```

## 2.24 Менеджер событий EventsManager

Цель данного компонента состоит в добавлении возможности перехватывать процесс выполнения большинства компонентов системы путём создания специальных “ключевых точек”. Эти ключевые точки

позволяют разработчику получить информацию о состоянии, манипулировать данными и изменять процесс работы компонента.

### 2.24.1 Пример использования

В следующем примере, мы используем менеджер событий для прослушивания событий вызываемых в MySQL соединении управляемым *Phalcon\Db*. Для начала нам необходимо создать объект слушателя. Методы класса являются событиями, которые необходимо прослушивать.

```
<?php

class MySqlListener
{

    public function afterConnect()
    {

    }

    public function beforeQuery()
    {

    }

    public function afterQuery()
    {

    }
}
```

Такой класс может реализовывать необходимые нам события. Менеджер событий будет взаимодействовать между компонентом и нашим классом, вызывая события, реализованные методами класса и поддерживаемые компонентом.

```
<?php

use Phalcon\Events\Manager as EventsManager,
    Phalcon\Db\Adapter\Pdo\Mysql as DbAdapter;

$eventsManager = new EventsManager();

// Создание слушателя базы данных
$dbListener = new MySqlListener();

// Слушать все события базы данных
$eventsManager->attach('db', $dbListener);

$connection = new DbAdapter(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "secret",
    "dbname" => "invo"
));

// Совмещение менеджера событий с адаптером базы данных
$connection->setEventsManager($eventsManager);
```

```
// Выполнение SQL запроса
$connection->query("SELECT * FROM products p WHERE p.status = 1");
```

Для того, чтобы получать все SQL-запросы, выполненные в нашем приложении, мы должны использовать событие “afterQuery”. Первый передаваемый слушателю параметр содержит контекстную информацию о текущем событии, второй параметр - само соединение.

```
<?php

use Phalcon\Logger\Adapter\File as Logger;

class MyDbListener
{

    protected $_logger;

    public function __construct()
    {
        $this->_logger = new Logger("../apps/logs/db.log");
    }

    public function afterQuery($event, $connection)
    {
        $this->_logger->log($connection->getSQLStatement(), \Phalcon\Logger::INFO);
    }

}
```

В рамках этого примера, мы будем также использовать профайлер Phalcon\Db\Profiler для обнаружения SQL-запросов с длительным временем выполнения:

```
<?php

use Phalcon\Db\Profiler,
    Phalcon\Logger,
    Phalcon\Logger\Adapter\File;

class MyDbListener
{

    protected $_profiler;

    protected $_logger;

    /**
     * Создаем профайлер и запускаем логгер
     */
    public function __construct()
    {
        $this->_profiler = new Profiler();
        $this->_logger = new Logger("../apps/logs/db.log");
    }

    /**
     * Этот метод будет запущен, если будет вызван метод 'beforeQuery'
     */
    public function beforeQuery($event, $connection)
    {
        $this->_profiler->startProfile($connection->getSQLStatement());
    }

}
```

```
}

/**
 * Этот метод будет запущен, если будет вызван метод 'afterQuery',
 */
public function afterQuery($event, $connection)
{
    $this->_logger->log($connection->getSQLStatement(), Logger::INFO);
    $this->_profiler->stopProfile();
}

public function getProfiler()
{
    return $this->_profiler;
}

}

Результатирующие данные о работе профайлера могут быть получены из слушателя:
```

```
<?php

// Выполнение SQL запроса
$connection->execute("SELECT * FROM products p WHERE p.status = 1");

foreach ($dbListener->getProfiler()->getProfiles() as $profile) {
    echo "SQL Statement: ", $profile->getSQLStatement(), "\n";
    echo "Start Time: ", $profile->getInitialTime(), "\n";
    echo "Final Time: ", $profile->getFinalTime(), "\n";
    echo "Total Elapsed Time: ", $profile->getTotalElapsedSeconds(), "\n";
}
```

Подобным образом мы можем зарегистрировать лямбда-функцию для выполнения этой задачи, без использования отдельного класса слушателя (как в примере выше):

```
<?php

// Слушаем все события базы данных
$eventManager->attach('db', function($event, $connection) {
    if ($event->getType() == 'afterQuery') {
        echo $connection->getSQLStatement();
    }
});
```

## 2.24.2 Создание компонентов с поддержкой событий

Компоненты, созданные в вашем приложении, могут инициировать события в EventsManager. Вы также можете создавать слушателей, которые реагируют на эти события. В следующем примере мы создаем компонент, под названием “MyComponent”. Этот компонент будет указывать менеджеру событий о выполнении своего метода “someTask”, что в свою очередь будет вызывать два события для слушателей в EventsManager:

```
<?php

use Phalcon\Events\EventsAwareInterface

class MyComponent implements EventsAwareInterface
```

```
{
    protected $_eventsManager;

    public function setEventsManager($eventsManager)
    {
        $this->_eventsManager = $eventsManager;
    }

    public function getEventsManager()
    {
        return $this->_eventsManager;
    }

    public function someTask()
    {
        $this->_eventsManager->fire("my-component:beforeSomeTask", $this);

        // тут выполнение каких-либо действий

        $this->_eventsManager->fire("my-component:afterSomeTask", $this);
    }
}
```

Обратите внимание, что события, создаваемые нашим компонентом, имеют префикс “my-component”. Это уникальное слово для разделения событий, которые формируются из разных компонентов. Вы можете создавать события вне компонента с таким же именем, оно ни от чего не зависит. Теперь давайте создадим слушателя для нашего компонента:

```
<?php

class SomeListener
{

    public function beforeSomeTask($event, $myComponent)
    {
        echo "Выполняется beforeSomeTask\n";
    }

    public function afterSomeTask($event, $myComponent)
    {
        echo "Выполняется afterSomeTask\n";
    }
}
```

Слушатель - это просто класс, который реализует все события, вызываемые в компоненте. Давайте заставим их работать вместе:

```
<?php

// Создаём менеджер событий
$eventsManager = new Phalcon\Events\Manager();

// Создаём экземпляр MyComponent
$myComponent = new MyComponent();

// Связываем компонент и менеджер событий
```

```
$myComponent->setEventManager($eventsManager);  
  
// Связываем слушателя и менеджер событий  
$eventsManager->attach('my-component', new SomeListener());  
  
// Выполняем метод нашего компонента  
$myComponent->someTask();
```

Когда метод “someTask” выполнится, сработают оба метода слушателя, и выведутся следующие строки:

```
Выполняется beforeSomeTask  
Выполняется afterSomeTask
```

Во время наступления события в слушателей можно передавать дополнительные данные, они должны передаваться третьим параметром в метод “fire”:

```
<?php  
  
$eventsManager->fire("my-component:afterSomeTask", $this, $extraData);
```

Слушатель также получает эти данные третьим параметром:

```
<?php  
  
// Получение данных из третьего параметра  
$eventManager->attach('my-component', function($event, $component, $data) {  
    print_r($data);  
});  
  
// Получение данных из контекста события  
$eventManager->attach('my-component', function($event, $component) {  
    print_r($event->getData());  
});
```

Если слушать необходимо только определённое событие, вы можете указать его в момент связывания:

```
<?php  
  
// Обработчик выполнится только при наступлении события "beforeSomeTask"  
$eventManager->attach('my-component:beforeSomeTask', function($event, $component) {  
    //...  
});
```

### 2.24.3 Остановка/Продолжение событий

Несколько слушателей может быть привязано к одному событию, это означает, что при его наступлении эти слушатели будут уведомлены. Слушатели уведомляются в порядке, в котором они были зарегистрированы в менеджере событий EventsManager. Некоторые события могут быть прекращены во время работы слушателя и уведомление других слушателей будет остановлено.

```
<?php  
  
$eventsManager->attach('db', function($event, $connection){  
  
    // Если событие поддерживает прекращение  
    if ($event->isCancelable()) {  
        // Прекращение события, остальные слушатели его не получат  
        $event->stop();  
    }  
});
```

```

}

//...

} ;

```

По умолчанию все события поддерживают прекращение, большинство событий, выполняемых в ядре фреймворка, тоже поддерживают прекращение. Вы можете указать, что событие не прекращаемое передавая “false” в четвертый параметр вызова fire:

```
<?php

$eventsManager->fire("my-component:afterSomeTask", $this, $extraData, false);
```

#### 2.24.4 Настройка слушателей (Listener)

При установке слушателей можно устанавливать их приоритет. Это позволяет указать порядок их вызова в момент выполнения.

```
<?php

// активация установки приоритетов
$evManager->enablePriorities(true);

$evManager->attach('db', new DbListener(), 150); // Высокий приоритет
$evManager->attach('db', new DbListener(), 100); // Нормальный приоритет
$evManager->attach('db', new DbListener(), 50); // Низкий приоритет
```

#### 2.24.5 Сбор ответов

Менеджер событий умеет собрать каждый ответ, возвращаемый каждым слушателем, пример ниже показывает как это можно использовать:

```
<?php

use Phalcon\Events\Manager as EventsManager;

$evManager = new EventsManager();

// Настройка сборщика ответов
$evManager->collectResponses(true);

// Добавления слушателя
$evManager->attach('custom:custom', function() {
    return 'first response';
});

// Добавления еще одного слушателя
$evManager->attach('custom:custom', function() {
    return 'second response';
});

// Выполнение события
$evManager->fire('custom:custom', null);
```

```
// Получаем все ответы
print_r($evManager->getResponses());
```

Сформируются такие данные:

```
Array ( [0] => first response [1] => second response )
```

## 2.24.6 Создание собственных менеджеров событий (EventsManager)

Для создания менеджера необходимо реализовать интерфейс *Phalcon\Events\ManagerInterface* и заменить им стандартный менеджер EventsManager при инициализации Phalcon.

## 2.25 Заголовки запроса (Request)

Каждый HTTP-запрос (исходящий как правило из браузера) содержит дополнительную информацию о запросе, такую как заголовок, данные, файлы, переменные и т.д. Веб-приложению требуется разобрать и проанализировать эту информацию, чтобы возвратить правильный ответ. *Phalcon\HTTP\Request* инкапсулирует информацию запроса, позволяя вам получать доступ к ней объектно-ориентированным способом.

```
<?php

// Получаем экземпляр объекта request
$request = new \Phalcon\Http\Request();

// Проверка что данные пришли методом POST
if ($request->isPost() == true) {

    // Проверка что request создан через Ajax
    if ($request->isAjax() == true) {
        echo "Request создан используя POST и AJAX";
    }
}
```

### 2.25.1 Получение значений

PHP автоматически заполняет суперглобальные массивы `$_GET` и `$_POST` в зависимости от типа запроса. Эти массивы содержат значения, которые получены из формы или параметры, отправляемые через URL. Переменные в массивах небезопасны и могут содержать недопустимые символы или даже вредоносный код, который может привести к SQL injection или Cross Site Scripting (XSS) атакам.

*Phalcon\HTTP\Request* предоставляет доступ к значениям `$_REQUEST`, `$_GET` и `$_POST` массивам и обезопасивает или фильтрует через специальный сервис ‘filter’, (по умолчанию *Phalcon\Filter*). Следующие примеры показывают одинаковое поведение:

```
<?php

// Ручная фильтрация
$filter = new Phalcon\Filter();

$email = $filter->sanitize($_POST["user_email"], "email");

// Ручная фильтрация значения
```

```

$filter = new Phalcon\Filter();
$email  = $filter->sanitize($request->getPost("user_email"), "email");

// Автоматическая фильтрация значения
$email = $request->getPost("user_email", "email");

// Получение значения по умолчанию, если параметр равен NULL
$email = $request->getPost("user_email", "email", "some@example.com");

// Получение значения по умолчанию, если параметр равен NULL без использования фильтрации
$email = $request->getPost("user_email", null, "some@example.com");

```

## 2.25.2 Доступ к Request из Контроллера

Доступ к Request чаще всего требуется в действиях контроллера. Для доступа к объекту *Phalcon\HTTP\Request* из контроллера, необходимо обратиться к публичному свойству `$this->request`:

```

<?php

use Phalcon\Mvc\Controller;

class PostsController extends Controller
{

    public function indexAction()
    {

    }

    public function saveAction()
    {

        // Проверка что данные пришли методом POST
        if ($this->request->isPost() == true) {

            // Получение POST данных
            $customerName = $this->request->getPost("name");
            $customerBorn = $this->request->getPost("born");

        }
    }
}

```

## 2.25.3 Загрузка файлов

Еще одна частая задача - загрузка файлов *Phalcon\HTTP\Request* предлагает объектно-ориентированный подход для решения этой задачи:

```

<?php

use Phalcon\Mvc\Controller;

class PostsController extends Controller

```

```
{  
  
    public function uploadAction()  
    {  
        // Проверяем что файл загружен  
        if ($this->request->hasFiles() == true) {  
  
            // Выводим имя и размер файла  
            foreach ($this->request->getUploadedFiles() as $file) {  
  
                // Выводим детали  
                echo $file->getName(), " ", $file->getSize(), "\n";  
  
                // Перемещаем в приложение  
                $file->moveTo('files/' . $file->getName());  
            }  
        }  
    }  
  
}
```

Каждый объект, возвращаемый `Phalcon\Http\Request::getUploadedFiles()` является экземпляром `Phalcon\Http\Request\File`. Использование суперглобального массива `$_FILES` предоставляет такое же поведение. `Phalcon\Http\Request\File` инкапсулирует только информацию, относящуюся к каждому загруженному в текущем запросе файлу.

## 2.25.4 Работа с заголовками

Как уже упоминалось выше, заголовки запросов содержат полезную информацию, которая позволит нам отправить правильный ответ пользователю. Следующие примеры показывают, как получить эту информацию:

```
<?php  
  
// Получение заголовка Http-X-Requested-With  
$requestedWith = $response->getHeader("X_REQUESTED_WITH");  
if ($requestedWith == "XMLHttpRequest") {  
    echo "Запрос отправлен через Ajax";  
}  
  
// Или так  
if ($request->isAjax()) {  
    echo "The request was made with Ajax";  
}  
  
// Проверка уровня запроса  
if ($request->isSecureRequest() == true) {  
    echo "The request was made using a secure layer";  
}  
  
// Получение IP сервера, например 192.168.0.100  
$ipAddress = $request->getServerAddress();  
  
// Получение IP клиента, например 201.245.53.51  
$ipAddress = $request->getClientAddress();  
  
// Получение строки User Agent (HTTP_USER_AGENT)
```

```
$userAgent = $request->getUserAgent();

// Получение оптимального типа контента для браузера, например text/xml
$contentType = $request->getAcceptableContent();

// Получение лучшей кодировки для браузера, например utf-8
$charset = $request->getBestCharset();

// Получение лучшего языка на который настроен браузер, например en-us
$language = $request->getBestLanguage();
```

## 2.26 Заголовки ответа (Responses)

Одной из частей работы HTTP-протокола является возвращение ответа клиенту. В Phalcon существует компонент `Phalcon\Http\Response` для реализации этой задачи. HTTP-ответ состоит из заголовков и тела ответа. Типичное использование Response выглядит следующим образом:

```
<?php

// Получение экземпляра Response
$response = new \Phalcon\Http\Response();

// Установка кода статуса
$response->setStatusCode(404, "Not Found");

// Установка содержимого ответа
$response->setContent("Сожалеем, но страница не существует");

// Отправка ответа клиенту
$response->send();
```

Имейте в виду, что при использовании полного стека MVC нет необходимости отправлять результаты Response вручную. Однако, если есть необходимость указать ответ самостоятельно в действии контроллера, то можно использовать такой пример:

```
<?php

class FeedController extends Phalcon\Mvc\Controller
{

    public function getAction()
    {
        // Получение экземпляра Response
        $response = new \Phalcon\Http\Response();

        $feed = //... тут данные

        // Установка содержимого ответа
        $response->setContent($feed->asString());

        // Возврат Response ответа
        return $response;
    }
}
```

## 2.26.1 Работа с заголовками

Заголовки являются важной частью для HTTP-ответов. Они содержат полезную информацию о статусе ответа, его типе и еще многое другое.

Указывать заголовки можно следующим образом:

```
<?php

// Установка по имени
$response->setHeader("Content-Type", "application/pdf");
$response->setHeader("Content-Disposition", 'attachment; filename="downloaded.pdf"');

// Установка напрямую
$response->setRawHeader("HTTP/1.1 200 OK");
```

Объект *Phalcon\HTTP\Response\Headers* содержит в себе все заголовки и средства для их управления. Этот класс позволяет управлять заголовками до их отправки клиенту:

```
<?php

// Получение всех заголовков
$headers = $response->getHeaders();

// Получение заголовка по имени
$contentType = $response->getHeaders()->get("Content-Type");
```

## 2.26.2 Создание перенаправлений (редиректы)

С помощью *Phalcon\HTTP\Response* вы можете выполнять переадресования HTTP:

```
<?php

// Переадресация на корневой URI
$response->redirect();

// Перенаправление на внутренний URI
$response->redirect("posts/index");

// Перенаправление на внешнюю ссылку
$response->redirect("http://en.wikipedia.org", true);

// Перенаправление со специальным HTTP-кодом
$response->redirect("http://www.example.com/new-location", true, 301);
```

Все ссылки обслуживаются внутренним сервисом ‘url’ (по умолчанию это *Phalcon\Mvc\Url*), в таком случае вы можете использовать перенаправления на определённые в приложении маршруты (роуты):

```
<?php

// Переадресация по именованному правилу роутинга
return $response->redirect(array(
    "for" => "index-lang",
    "lang" => "jp",
    "controller" => "index"
));
```

Обратите внимание, что при создании перенаправления не отключается компонент отображения (Views), так что действие, в котором оно вызывается, всё равно будет выполнено. Вы можете отключить отображение из контроллера, выполнив `$this->view->disable();`

### 2.26.3 HTTP-кэширование

Одним из самых простых способов повышения производительности приложения является снижение трафика с помощью HTTP-кэширования. Большинство современных браузеров поддерживают HTTP-кэширование и это является одной из причин, почему многие веб-сайты в настоящее время работают достаточно быстро.

Секретные заголовки отправляемые при первой передаче страницы:

- *Expires*: Устанавливая этот заголовок в прошлое или будущее можно указывать браузуру срок жизни страницы.
- *Cache-Control*: Позволяет указать сколько времени страница должна считаться для браузера актуальной.
- *Last-Modified*: Указывает браузеру когда было последнее изменение страницы, что позволяет избежать повторной загрузки страницы.
- *ETag*: Представляет собой уникальный идентификатор, который должен быть сформирован с учетом времени изменения текущей страницы.

#### Expires

Указание срока жизни является одним из наиболее удобных и эффективных способов кэширования страниц на стороне клиента (браузера). Мы добавим дополнительный срок к текущему времени, это укажет браузеру сохранять страницу в кэше пока этот срок не истечет и не обращаться за ней к серверу:

```
<?php

$expireDate = new DateTime();
$expireDate->modify('+2 months');

$response->setExpires($expireDate);
```

Ответ в компоненте Response автоматически преобразует дату для временной зоны GMT, именно так как ожидается в заголовке Expires.

Более того, если мы укажем прошедшую дату, то это указывает браузеру всегда обновлять запрошенную страницу:

```
<?php

$expireDate = new DateTime();
$expireDate->modify('-10 minutes');

$response->setExpires($expireDate);
```

Браузеры основываются на системных часах клиента для определения наступления этой даты. Так как часы на клиенте могут быть изменены, то срок жизни будет некорректен. Это ограничение такого механизма кэширования.

## Cache-Control

Этот заголовок осуществляет более безопасный способ кэширования. Мы просто указываем браузеру время в секундах которое необходимо хранить страницы в кэше:

```
<?php  
  
// кэшировать на сутки с текущего момента  
$response->setHeader('Cache-Control', 'max-age=86400');
```

Противоположный эффект (для запрета кэширования страницы) организуется следующим образом:

```
<?php  
  
// Не кэшировать  
$response->setHeader('Cache-Control', 'private, max-age=0, must-revalidate');
```

## E-Tag

Заголовок “entity-tag” или кратко “E-tag” позволяет браузеру понять, была ли изменена страница между двумя запросами. Идентификатор должен рассчитываться таким образом, что бы измениться если изменено содержимое страницы:

```
<?php  
  
// Формирование значения E-Tag основанное на последнем времени изменения новости  
$recentDate = News::maximum(array('column' => 'created_at'));  
$eTag = md5($recentDate);  
  
// Отправка E-Tag  
$response->setHeader('E-Tag', $eTag);
```

## 2.27 Управление Кукиами

Куки очень полезный способ хранения маленьких фрагментов данных на стороне клиента, которые могут быть получены, даже если пользователь закроет свой браузер. [Phalcon](#) | [Http](#) | [Response](#) | [Cookies](#) выступает в качестве глобального “мешка” (bag) для Куки. Куки хранятся в таком “мешке” (bag) во время выполнения запроса и отправляются автоматически по его окончанию.

### 2.27.1 Базовое использование

Вы можете установить или извлечь Куки простым обращением к сервису ‘cookies’ из любого места в приложении:

```
<?php  
  
class SessionController extends Phalcon\Mvc\Controller  
{  
    public function loginAction()  
    {  
        // Проверяем была ли установлена Кука ранее  
        if ($this->cookies->has('remember-me')) {
```

```

    // Извлекаем Куку
    $rememberMe = $this->cookies->get('remember-me');

    // Извлекаем значение из Куки
    $value = $rememberMe->getValue();

}

}

public function startAction()
{
    $this->cookies->set('remember-me', 'некоторое значение', time() + 15 * 86400);
}
}

```

## 2.27.2 Шифрование/десифрование Кука

По умолчанию Куки автоматически шифруются перед отправкой клиенту и расшифровываются при получении. Такая защита не позволяет неавторизированным пользователям видеть содержимое Кука на стороне клиента (в браузере), но несмотря на это, хранить в них конфиденциальные (персональные) данные не следует.

Вы можете отключить шифрование следующим образом:

```
<?php

$di->set('cookies', function() {
    $cookies = new Phalcon\Http\Response\Cookies();
    $cookies->useEncryption(false);
    return $cookies;
});
```

При использовании шифрования должен быть установлен глобальный ключ в сервисе ‘сгурт’:

```
<?php

$di->set('crypt', function() {
    $crypt = new Phalcon\Crypt();
    $crypt->setKey('#1dj8$=dp?.ak//j1V$'); // Используйте свой собственный ключ!
    return $crypt;
});
```

Отправка клиентам в куки без шифрования объектов со сложной структурой, наборы результатов, служебную информацию и другую подобную информацию, может раскрыть детали реализации приложения, которыми могут воспользоваться злоумышленники для взлома вашего приложения. Если вы не хотите использовать шифрование, мы настоятельно рекомендуем вам отправлять только очень простые данные, такие как числа и небольшие строки.

## 2.28 Генерация ссылок (URLs)

Компонент *Phalcon\ Mvc\ Url* позволяет генерировать ссылки для приложений Phalcon. Он может формировать ссылки основываясь на маршрутах.

## 2.28.1 Указание базового URI

В зависимости от корня установленного приложение, может появиться необходимость указания базового URL.

Например, если корневой каталог `/var/www/htdocs`, а ваше приложение установлено в `/var/www/htdocs/invo`, базовый URI (`baseUri`) будет “`/invo/`”. При использовании виртуальных хостов, или приложение установлено в корневой каталог, параметр `baseUri` будет “`/`”. Для выявления какой адрес Phalcon считает за `baseUri` можно выполнить такой сценарий:

```
<?php  
  
$url = new Phalcon\Mvc\Url();  
echo $url->getBaseUri();
```

По умолчанию Phalcon самостоятельно выявляет необходимый `baseUri`, но в целях повышения производительности советуем указать его вручную:

```
<?php  
  
$url = new Phalcon\Mvc\Url();  
  
// Относительный URI  
$url->setBaseUri('/invo/');  
  
// Установка доменного имени  
$url->setBaseUri('://my.domain.com/');  
  
// Установка корневой полной ссылки  
$url->setBaseUri('http://my.domain.com/my-app/');
```

Компонент, как правило, регистрируется в DI-контейнере, что позволяет его легко настроить:

```
<?php  
  
$di->set('url', function(){  
    $url = new Phalcon\Mvc\Url();  
    $url->setBaseUri('/invo/');  
    return $url;  
});
```

## 2.28.2 Генерация ссылок

По умолчанию для создания ссылок используется компонент `Router`. Ваше приложение может работать используя по умолчанию такую схему маршрутизации: `/:controller/:action/:params`. Соответственно, легко создавать ссылки по этому образцу (или другим правилам, прописанным в маршрутизаторе) передавая параметры в метод “`get`”:

```
<?php echo $url->get("products/save") ?>
```

Обратите внимание: указывать базовый URL нет необходимости. При использовании именованных маршрутов ссылки можно формировать динамически. Например, у вас есть такой маршрут:

```
<?php  
  
$route->add('/blog/{year}/{month}/{title}', array(  
    'controller' => 'posts',
```

```
'action' => 'show',
))->setName('show-post');
```

Ссылку на него можно сформировать таким образом:

```
<?php

// Получимся: /blog/2012/01/some-blog-post
$url->get(array(
    'for' => 'show-post',
    'year' => 2012,
    'month' => '01',
    'title' => 'some-blog-post'
));
```

### 2.28.3 Создание ссылок без Mod-Rewrite

Компонент можно использовать для создания ссылок без mod-rewrite:

```
<?php

$url = new Phalcon\Mvc\Url();

// Указание базового адреса из $_GET["_url"]
$url->setBaseUri('/invo/index.php?url=/');

// Получимся: /invo/index.php?url=/products/save
echo $url->get("products/save");
```

Вы так же можете использовать `$_SERVER["REQUEST_URI"]`:

```
<?php

$url = new Phalcon\Mvc\Url();

// Указание базового адреса используя $_GET["_url"]
$url->setBaseUri('/invo/index.php?url=/');

// Передача URI из $_SERVER["REQUEST_URI"]
$url->setBaseUri('/invo/index.php/');
```

В таком случае необходимо самостоятельно передать URI для обработки в Router:

```
<?php

$router = new Phalcon\Mvc\Router();

// ... указание правил маршрутизации

$uri = str_replace($_SERVER["SCRIPT_NAME"], '', $_SERVER["REQUEST_URI"]);
$router->handle($uri);
```

Получится маршрут:

```
<?php

// Будет сформировано: /invo/index.php/products/save
echo $url->get("products/save");
```

## 2.28.4 Создание ссылок в Volt

Функция “url”, доступная в Volt, позволяет формировать ссылки с использованием этого компонента:

```
<a href="{{ url('posts/edit/1002') }}">Редактировать</a>
```

Генерация статических маршрутов:

```
<link rel="stylesheet" href="{{ static_url('css/style.css') }}" type="text/css" />
```

## 2.28.5 Статические против динамических URI

Этот компонент позволит вам настроить другой базовый URI для статических ресурсов в приложении:

```
<?php  
  
$url = new Phalcon\Mvc\Url();  
  
// Динамический URI  
$url->setBaseUri('/');  
  
// Статические ресурсы проходят через CDN  
$url->setStaticBaseUri('http://static.example.com/');
```

*Phalcon* Tag будет запрашивать как динамические, так и статические URI, используя этот компонент.

## 2.28.6 Реализация своего генератора ссылок

Для создания собственного генератора необходимо реализовать интерфейс *Phalcon\ Mvc\ UrlInterface*, или использовать наследование и переопределить стандартный компонент Phalcon.

## 2.29 Информационные сообщения

Информационные сообщения используются для уведомления пользователей о состоянии выполненных действий или просто показывают необходимую информацию. Такой тип сообщений может быть сгенерирован с помощью этого компонента.

### 2.29.1 Адаптеры

Этот компонент включает в себя несколько адаптеров, чтобы определить поведение сообщения:

Адаптер	Описание	API
Direct Session	Выводит сообщение напрямую пользователю Временно сохраняет сообщения в сессию для вывода в следующем запросе	<i>Phalcon\Flash\Direct</i> <i>Phalcon\Flash\Session</i>

## 2.29.2 Использование

Обычно компонент всплывающих сообщений доступен из контейнера сервисов. Если вы используете `Phalcon\DI\FactoryDefault`, то `Phalcon\Flash\Direct` будет автоматически зарегистрирован как “flash” сервис:

```
<?php

// Устанавливаем сервис
$di->set('flash', function() {
    return new \Phalcon\Flash\Direct();
});
```

Таким образом, вы можете использовать его в контроллерах или в представлениях (view):

```
<?php

class PostsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function saveAction()
    {
        $this->flash->success("The post was correctly saved!");
    }

}
```

Существует четыре типа встроенных сообщений:

```
<?php

$this->flash->error("too bad! the form had errors");
$this->flash->success("yes!, everything went very smoothly");
$this->flash->notice("this a very important information");
$this->flash->warning("best check yo self, you're not looking too good.");
```

Вы можете добавлять сообщения со своими типами:

```
<?php

$this->flash->message("debug", "this is debug message, you don't say");
```

## 2.29.3 Вывод сообщений

Сообщения, посланные в компонент, автоматически форматируются с html:

```
<div class="errorMessage">too bad! the form had errors</div>
<div class="successMessage">yes!, everything went very smoothly</div>
<div class="noticeMessage">this a very important information</div>
<div class="warningMessage">best check yo self, you're not looking too good.</div>
```

Как видно на примере выше - используются некоторые CSS классы, которые автоматически добавляются в тег ‘DIV’. Эти классы позволяют вам видоизменять вывод сообщений. CSS классы могут быть

изменены, например, если вы используете Twitter Bootstrap, то можно указать следующие классы:

```
<?php

// Регистрируем компонент сообщений с CSS классами
$di->set('flash', function(){
    $flash = new \Phalcon\Flash\Direct(array(
        'error' => 'alert alert-error',
        'success' => 'alert alert-success',
        'notice' => 'alert alert-info',
    ));
    return $flash;
});
```

После этого сообщения будут выводиться таким образом:

```
<div class="alert alert-error">too bad! the form had errors</div>
<div class="alert alert-success">yes!, everything went very smoothly</div>
<div class="alert alert-info">this a very important information</div>
```

## 2.29.4 Понимание разницы между адаптерами Direct и Session

В зависимости от адаптера, используемого для отправки сообщений, вывод будет производиться сразу или временно сохраняться в сессии для дальнейшего вывода. В каких случаях надо их использовать? Это обычно зависит от типа перенаправления, которое вы делаете после отправки сообщения. Например, если вы делаете прямой вывод (или внутреннее перенаправление), то сохранять в сессии нет необходимости, но если вы делаете HTTP-перенаправление, то сообщения необходимо сохранить в сессии, чтобы их можно было позже вывести пользователю:

```
<?php

class ContactController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function saveAction()
    {

        // Сохраняем объект в БД

        // Выводим прямое сообщение
        $this->flash->success("Your information were stored correctly!");

        // Делаем внутреннее перенаправление на другое действие
        return $this->dispatcher->forward(array("action" => "index"));
    }
}
```

Или используя HTTP-перенаправление:

```
<?php

class ContactController extends \Phalcon\Mvc\Controller
```

```

{
    public function indexAction()
    {
        }

    public function saveAction()
    {
        // Сохраняем объект в БД

        // Отправляем сообщение в сессию
        $this->flashSession->success("Your information were stored correctly!");

        // Делаем полное HTTP-перенаправление
        return $this->response->redirect("contact/index");
    }
}

```

В таком случае вам необходимо вручную вывести сообщение в соответствующем представлении:

```

<!-- app/views/contact/index.phtml -->

<p><?php $this->flashSession->output() ?></p>

```

Атрибут ‘flashSession’ означает, каким способом изначально был задан компонент в контейнере сервисов. Вам необходимо запустить *сессии*, чтобы успешно использовать такой тип сообщений.

## 2.30 Сохранение данных в сессии

Компонент Phalcon\Session предоставляет объектно-ориентированный интерфейс для работы с сессиями.

Причины использования этого компонента, а не обычных сессий:

- Вы можете легко изолировать сессии данных в различных приложениях на одном домене
- Можно перехватить места установки/получения данных в приложении
- Использование адаптера сессий, оптимального для текущего приложения

### 2.30.1 Запуск сессий

Некоторые приложения активно используют в своей работе сессии, используя их в каждом действии. Другие наоборот, используют сессии мало и не часто. Благодаря использованию контейнера сервисов, мы можем гарантировать, что запуск сессий будет произведён только по необходимости.

```

<?php

// Сессии запускаются один раз, при первом обращении к объекту
$di->setShared('session', function() {
    $session = new Phalcon\Session\Adapter\Files();
    $session->start();
});

```

```
        return $session;
});
```

### 2.30.2 Сохранение/получение данных из сессий

Из контроллера, представления (view) или другого компонента расширяющего *Phalcon\DI\Injectable* можно получить доступ к сессиям и работать с ними следующим образом:

```
<?php

class UserController extends Phalcon\Mvc\Controller
{

    public function indexAction()
    {
        // Установка значения сессии
        $this->session->set("user-name", "Michael");
    }

    public function welcomeAction()
    {

        // Проверка наличия переменной сессии
        if ($this->session->has("user-name")) {

            // Получение значения
            $name = $this->session->get("user-name");
        }
    }
}
```

### 2.30.3 Удаление/очистка сессий

Таким же способом можно удалить переменную сессии, или целиком очистить сессию:

```
<?php

class UserController extends Phalcon\Mvc\Controller
{

    public function removeAction()
    {
        // Удаление переменной сессии
        $this->session->remove("user-name");
    }

    public function logoutAction()
    {
        // Полная очистка сессии
        $this->session->destroy();
    }
}
```

## 2.30.4 Изоляция данных сессии внутри приложения

Иногда пользователь может запускать одно и тоже приложение несколько раз, на одном и том же сервере, в одно время. Естественно, используя переменные сессий нам бы хотелось, чтобы все приложения получали доступ к разным сессиям (хотя в одинаковых приложениях и код одинаковый и названия переменных). Для решения этой проблемы можно использовать префикс для переменных сессий, разный для разных приложений.

```
<?php

// Изоляция данных сессий
$di->set('session', function(){

    // Все переменные этого приложения будет иметь префикс "my-app-1"
    $session = new Phalcon\Session\Adapter\Files(
        array(
            'uniqueId' => 'my-app-1'
        )
    );

    $session->start();

    return $session;
});


```

На работе это никак не скажется. Добавлять префикс вручную во время установки или чтения сессий нет необходимости.

## 2.30.5 Наборы сессий (Session Bags)

Компонент *Phalcon\Session\Bag* (Session Bags, дословно “Мешки с сессиями”) позволяет работать с сессиями разделяя их по пространствам имён. Работая таким образом, вы можете легко создавать группы переменных сессии в приложении. Установив значение переменной такого объекта, оно автоматически сохранится в сессии:

```
<?php

$user      = new Phalcon\Session\Bag('user');
$user->setDI($di);
$user->name = "Kimbra Johnson";
$user->age  = 22;
```

## 2.30.6 Сохранение данных в компонентах

Контроллеры, компоненты и классы расширяющие *Phalcon\DI\Injectable* могут работать с *Phalcon\Session\Bag* напрямую. Компонент в таком случае изолирует данные для каждого класса. Благодаря этому вы можете сохранять данные между запросами, используя их как обычные переменные.

```
<?php

class UserController extends Phalcon\Mvc\Controller
{

    public function indexAction()
```

```
{  
    // Создаётся постоянная (persistent) переменная "name"  
    $this->persistent->name = "Laura";  
}  
  
public function welcomeAction()  
{  
    if (isset($this->persistent->name))  
    {  
        echo "Привет, ", $this->persistent->name;  
    }  
}  
}
```

И в компоненте:

```
<?php  
  
class Security extends Phalcon\Mvc\User\Component  
{  
  
    public function auth()  
    {  
        // Создаётся постоянная (persistent) переменная "name"  
        $this->persistent->name = "Laura";  
    }  
  
    public function getAuthName()  
    {  
        return $this->persistent->name;  
    }  
}
```

Данные, добавленные непосредственно в сессию (`$this->session`) доступны во всём приложении, в то время как persistent (`$this->persistent`) переменные доступны только внутри своего текущего класса.

### Реализация собственных адаптеров сессий

Для создания адаптера необходимо реализовать интерфейс `Phalcon\Session\AdapterInterface`, или использовать наследование от готового с доработкой необходимой логики.

У нас есть некоторые готовые адаптеры для сессий `Phalcon Incubator`

## 2.31 Фильтрация и очистка

Фильтрация пользовательского ввода является критической частью разработки приложений. Доверие или пренебрежение очисткой и фильтрацией ввода произведенного пользователем может привести к несанкционированному доступу к контенту вашего приложения, данным или даже к серверу, где ваше приложение размещено.

Полное изображение (с сайта xkcd)

Компонент `Phalcon\Filter` предоставляет набор основных фильтров для корректирования данных. Он обеспечивает объектно-ориентированную обертку вокруг PHP-фильтра.



### 2.31.1 Очистка данных

Очистка - это процесс, который удаляет определенный символ из значения, которое было введено пользователем. С помощью очистки входных данных мы гарантируем, что целостность приложения не будет нарушена.

```
<?php
```

```
$filter = new \Phalcon\Filter();

// возвращаем "someone@example.com"
$filter->sanitize("some(one)@exa\mple.com", "email");

// возвращает "hello"
$filter->sanitize("hello<<", "string");

// возвращает "100019"
$filter->sanitize("!100a019", "int");

// возвращает "100019.01"
$filter->sanitize("!100a019.01a", "float");
```

### 2.31.2 Очистка из контроллеров

Доступ к объекту `Phalcon\Filter` можно получить из контроллера для очистки GET или POST входных данных. Первым параметром является название переменной, которую необходимо получить, а вторым - название фильтра, который должен быть применен относительно этой переменной.

```
<?php
```

```
class ProductsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function saveAction()
    {

        // Чистим price из ввода
    }
}
```

```
$price = $this->request->getPost("price", "double");

// Чистим email из ввода
$email = $this->request->getPost("customerEmail", "email");

}

}
```

### 2.31.3 Фильтруем параметры действия (Action)

Следующий пример показывает, как чистить параметры действий в контроллере:

```
<?php

class ProductsController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function showAction($productId)
    {
        $productId = $this->filter->sanitize($productId, "int");
    }

}
```

### 2.31.4 Фильтрация данных

В дополнение к очистке, класс *Phalcon\Filter* так же предоставляет фильтрацию, которая изменяет или удаляет данные в соответствии с ожидаемым форматом.

```
<?php

$filter = new \Phalcon\Filter();

// возвращает "Hello"
$filter->sanitize("<hi>Hello</hi>", "striptags");

// возвращает "Hello"
$filter->sanitize(" Hello ", "trim");
```

### 2.31.5 Типы встроенных фильтров

В таблице приведены все типы фильтров, которыми располагает компонент:

Название	Описание
string	Преобразовывает теги
email	Удаляет все символы, за исключением букв, цифр и !#\$%&*+-/=?^_{' }~@.[].
int	Удаляет все символы, за исключением цифр и знаков плюс/минус.
float	Удаляет все символы, за исключением цифр, точек и знаков плюс/минус.
alphanum	Удаляет все символы, за исключением [a-zA-Z0-9]
striptags	Применяет <code>strip_tags</code> функцию
trim	Применяет <code>trim</code> функцию
lower	Применяет <code>strtolower</code> функцию
upper	Применяет <code>strtoupper</code> функцию

### 2.31.6 Создание собственных фильтров

Вы можете добавлять свои фильтры в `Phalcon\Filter`. Функция фильтрации может быть анонимной:

```
<?php

$filter = new \Phalcon\Filter();

// Используем анонимную функцию
$filter->add('md5', function($value) {
    return preg_replace('/[^0-9a-f]/', '', $value);
});

// Используем "md5" фильтр
$filtered = $filter->sanitize($possibleMd5, "md5");
```

Вы можете реализовать фильтр с помощью класса:

```
<?php

class IPv4Filter
{

    public function filter($value)
    {
        return filter_var($value, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4);
    }
}

$filter = new \Phalcon\Filter();

// Используем объект
$filter->add('ipv4', new IPv4Filter());

// Фильтруем с помощью "ipv4"
$filteredIp = $filter->sanitize("127.0.0.1", "ipv4");
```

### 2.31.7 Сложная очистка и фильтрация

PHP предоставляет отличную фильтрацию, которой вы можете воспользоваться. Посмотрите на документацию: [Фильтрация данных в документации PHP](#)

### 2.31.8 Разработка собственной системы фильтрации

Используйте интерфейс `Phalcon\FilterInterface` для создания собственной системы фильтрации, чтобы заменить существующую в Phalcon.

## 2.32 Контекстное экранирование

Веб-сайты и веб приложения уязвимы к XSS-атакам, несмотря на то, что PHP предоставляет некоторое количество функций для экранирования, в некоторых случаях этого бывает недостаточно. Класс `Phalcon\Escaper` предоставляет контекстное экранирование, позволяя выполнять экранированию любых текстов с минимальными затратами ресурсов.

Мы разработали компонент, базирующийся на XSS (Cross Site Scripting) Prevention Cheat Sheet, созданный в OWASP

Этот компонент полагается на mbstring для того, чтобы поддерживать любую кодировку.

Для демонстрации работы компонента и его важности рассмотрим следующий пример:

```
<?php

// Заголовок документа с вредоносным кодом
$maliciousTitle = '</title><script>alert(1)</script>';

// Вредоносные название CSS класса
$className = ';'_(';

// Вредоносное название CSS шрифта
$fontName = 'Verdana"</style>';

// Вредоносный Javascript текст
$javascriptText = "';</script>Hello";

// Создаем компонент экранирования
$e = new Phalcon\Escaper();

?>

<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

    <title><?php echo $e->escapeHtml($maliciousTitle) ?></title>

    <style type="text/css">
        .<?php echo $e->escapeCss($className) ?> {
            font-family : "<?php echo $e->escapeCss($fontName) ?>";
            color: red;
        }
    </style>
</head>

<body>

    <div class='<?php echo $e->escapeHtmlAttr($className) ?>'>hello</div>
```

```
<script>var some = 'php echo $e-&gt;escapeJs($javascriptText) ?&gt;'&lt;/script&gt;

&lt;/body&gt;
&lt;/html&gt;</pre

```

В итоге получим такой html документ:

```

1
2
3     <html>
4     <head>
5         <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
6
7         <title>&lt;/title&gt;&lt;script&gt;alert(1)&lt;/script&gt;</title>
8
9         <style type="text/css">
10            .\3c \2f style\3e {
11                font-family : "Verdana\22 \3c \2f style\3e ";
12                color: red;
13            }
14        </style>
15
16    </head>
17
18    <body>
19
20        <div class='&#x3c \2f style\3e '>hello</div>
21
22        <script>var some = '\x27\x3b\x3c\x2fscript\x3eHello'</script>
23
24    </body>
25
26    </html>

```

Все текстовые переменные были экранированы в соответствии с их контекстом. Использование необходимого контекста важно во избежания XSS-атак.

### 2.32.1 Экранирование HTML

Наиболее распространенная ситуация при вставке небезопасных данных между HTML-тегами:

```
<div class="comments"><!-- Экранируем данные, которым не доверяем! --></div>
```

Вы можете экранировать эти данные с помощью метода escapeHtml:

```
<div class="comments"><?php echo $e->escapeHtml('></div><h1>myattack</h1>'); ?></div>
```

Что приведет к:

```
<div class="comments">&gt;&lt;/div&gt;&lt;h1&gt;myattack&lt;/h1&gt;</div>
```

## 2.32.2 Экранирование HTML-атрибутов

Экранирование HTML-атрибутов отличается от простого экранирования HTML-контента. Экранирование изменяет все символы, не являющиеся буквами или цифрами. Этот вид экранирования предназначен для самых простых атрибутов, без учета сложных, таких как 'href' или 'url':

```
<table width="Экранируем данные, которым не доверяем!"><tr><td>Привет</td></tr></table>
```

Вы можете экранировать HTML-атрибуты, используя метод escapeHtmlAttr:

```
<table width="<?php echo $e->escapeHtmlAttr('"><h1>Привет</table') ; ?>"><tr><td>Привет</td></tr></table>
```

Что приведет к:

```
<table width=""><h1>Hello</table"><tr><td>Привет</td></tr></table>
```

## 2.32.3 Экранирование ссылок

Некоторые атрибуты, такие как 'href' или 'url' необходимо экранировать по-другому:

```
<a href="Экранируем данные, которым не доверяем!">Some link</a>
```

Вы можете экранировать этот HTML-атрибут, используя метод escapeUrl:

```
<a href="<?php echo $e->escapeUrl('"><script>alert(1)</script><a href="#">Ссылка</a>
```

Что приведет к:

```
<a href="%22%3E%3Cscript%3Ealert%281%29%3C%2Fscript%3E%3Ca%20href%3D%22%23">Ссылка</a>
```

## 2.32.4 Экранирование CSS

CSS идентификаторы/значения также могут быть экранированы:

```
<a style="color: Экранируем данные, которым не доверяем!">Ссылка</a>
```

Экранирование в этом случае можно выполнить с помощью метода escapeCss:

```
<a style="color: <?php echo $e->escapeCss('"><script>alert(1)</script><a href="#">Ссылка</a>
```

Что приведет к:

```
<a style="color: \22 \3e \3c script\3e alert\28 1\29 \3c \2f script\3e \3c a\20 href\3d \22 \23 ">Ссылка</a>
```

## 2.32.5 Экранирование Javascript

Строки, которые попадают в код javascript, тоже должны быть правильно экранированы:

```
<script>document.title = 'Экранируем данные, которым не доверяем!';</script>
```

Для этого используем метод escapeJs:

```
<script>document.title = '<?php echo $e->escapejs("'; alert(100); var x='"); ?>'</script>
```

```
<script>document.title = '\x27; alert(100); var x\x3d\x27;</script>
```

## 2.33 Валидация

Компонент PhalconValidation реализует независимую возможность проверки произвольного набора данных. Компонент можно использовать для проверки данных не относящихся к моделям или коллекциям.

Ниже показан пример использования компонента:

```
<?php

use Phalcon\Validation\Validator\PresenceOf,
    Phalcon\Validation\Validator\Email;

$validation = new Phalcon\Validation();

$validation->add('name', new PresenceOf(array(
    'message' => 'The name is required'
)));

$validation->add('email', new PresenceOf(array(
    'message' => 'The e-mail is required'
)));

$validation->add('email', new Email(array(
    'message' => 'The e-mail is not valid'
)));

$messages = $validation->validate($_POST);
if (count($messages)) {
    foreach ($messages as $message) {
        echo $message, '  
';
    }
}
```

Свободная блочная конструкция этого компонента позволяет вам создавать свои собственные валидаторы наряду с теми, что предоставляет фреймворк.

### 2.33.1 Инициализация валидации

Как вы уже поняли, цепочка проверок может быть инициализирована простым добавлением их в объект Phalcon\Validation. Вы можете повторно использовать код или лучше организовывать проверки, размещая их в отдельных файлах:

```
<?php

use Phalcon\Validation,
    Phalcon\Validation\Validator\PresenceOf,
    Phalcon\Validation\Validator\Email;

class MyValidation extends Validation
{
    public function initialize()
    {
```

```
$this->add('name', new PresenceOf(array(
    'message' => 'The name is required'
)));

$this->add('email', new PresenceOf(array(
    'message' => 'The e-mail is required'
)));

$this->add('email', new Email(array(
    'message' => 'The e-mail is not valid'
)));
}

}

<?php

$validation = new MyValidation();

$messages = $validation->validate($_POST);
if (count($messages)) {
    foreach ($messages as $message) {
        echo $message, '<br>';
    }
}
```

## 2.33.2 Валидаторы

Базовый компонент валидации Phalcon предоставляет следующие правила проверки:

Дополнительные проверки могут быть реализованы самостоятельно. Следующий класс объясняет, как создать правило валидации для этого компонента:

```
<?php

use Phalcon\Validation\Validator,
    Phalcon\Validation\ValidatorInterface,
    Phalcon\Validation\Message;

class IpValidator extends Validator implements ValidatorInterface
{

    /**
     * Выполнение валидации
     *
     * @param Phalcon\Validation $validator
     * @param string $attribute
     * @return boolean
     */
    public function validate($validator, $attribute)
    {
        $value = $validator->getValue($attribute);

        if (filter_var($value, FILTER_VALIDATE_IP, FILTER_FLAG_IPV4 | FILTER_FLAG_IPV6)) {

            $message = $this->getOption('message');
            if (!$message) {
                $message = 'IP адрес не правилен';
            }
        } else {
            $message = $this->getOption('message');
            if (!$message) {
                $message = 'IP адрес не правилен';
            }
        }

        if ($message) {
            $validator->addError($attribute, $message);
        }
    }
}
```

```

        }

        $validator->appendMessage(new Message($message, $attribute, 'Ip'));

        return false;
    }

    return true;
}

}

```

Важно помнить, что валидаторы возвращают булево значение, показывающее, прошла валидация успешно, либо нет.

### 2.33.3 Сообщения валидации

Компонент *Phalcon|Validation* имеет внутреннюю подсистему работы с сообщениями. Она обеспечивает гибкую работу с хранением и выводом проверочных сообщений, генерируемых в ходе проверки.

Каждое сообщение состоит из экземпляра класса *Phalcon|Validation\Message*. Набор сгенерированных сообщений может быть получен с помощью метода `getMessages()`. Каждое сообщение содержит расширенную информацию - атрибут, текст и тип сообщения:

```
<?php

$messages = $validation->validate();
if (count($messages)) {
    foreach ($validation->getMessages() as $message) {
        echo "Сообщение: ", $message->getMessage(), "\n";
        echo "Поле: ", $message->getField(), "\n";
        echo "Тип: ", $message->getType(), "\n";
    }
}

```

Метод `getMessages()` может быть переопределен в наследующем классе для замены/перевода текста сообщения по умолчанию, это особенно актуально для автоматически создаваемых валидаторов:

```
<?php

class MyValidation extends Phalcon\Validation
{

    public function initialize()
    {
        // ...
    }

    public function getMessages()
    {
        $messages = array();
        foreach (parent::getMessages() as $message) {
            switch ($message->getType()) {
                case 'PresenceOf':
                    $messages[] = 'Заполнение поля ' . $message->getField() . ' обязательно';
                    break;
            }
        }
    }
}
```

```
        return $messages;
    }
}
```

Или вы можете передать сообщение параметром по умолчанию в каждый валидатор:

```
<?php

use Phalcon\Validation\Validator\Email;

$validation->add('email', new Email(array(
    'message' => 'The e-mail is not valid'
))');
```

По умолчанию метод ‘getMessages’ возвращает все сообщения сгенерированные валидатором. Вы можете отфильтровать сообщения используя ‘filter’:

```
<?php

$messages = $validation->validate();
if (count($messages)) {
    // Отфильтровать только те сообщения, которые были сгенерированы для поля 'name'
    foreach ($validation->getMessages()->filter('name') as $message) {
        echo $message;
    }
}
```

## 2.33.4 Фильтрация данных

Данные фильтруются для того, чтобы быть уверенным, что вредоносные или неверные данные не будут пропущены приложением.

```
<?php

$validation = new Phalcon\Validation();

$validation
    ->add('name', new PresenceOf(array(
        'message' => 'The name is required'
    )));
    ->add('email', new PresenceOf(array(
        'message' => 'The email is required'
    )));
}

// Избавимся от лишних пробелов
$validation->setFilters('name', 'trim');
$validation->setFilters('email', 'trim');
```

Фильтрация и очистка производятся с помощью компонента *filter*. Вы можете добавлять в него свои фильтры, либо пользоваться встроенными.

## 2.33.5 События валидации

Когда в классах определена валидация, вы также можете реализовать методы ‘beforeValidation’ и ‘afterValidation’, чтобы добавить дополнительные проверки, очистку и т.п. Если ‘beforeValidation’ возвращает ‘false’, валидация не будет пройдена:

```
<?php

use Phalcon\Validation;

class LoginValidation extends Validation
{

    public function initialize()
    {
        // ...
    }

    /**
     * Выполняется перед валидацией
     *
     * @param array $data
     * @param object $entity
     * @param Phalcon\Validation\Message\Group $messages
     */
    public function beforeValidation($data, $entity, $messages)
    {
        if ($this->request->getHttpHost() != 'admin.mydomain.com') {
            $messages->appendMessage(new Message('Users only can log on in the administration domain'));
            return false;
        }
        return true;
    }

    /**
     * Выполняется после валидации
     *
     * @param array $data
     * @param object $entity
     * @param Phalcon\Validation\Message\Group $messages
     */
    public function afterValidation($data, $entity, $messages)
    {
        //... добавляем дополнительные сообщения или валидацию
    }
}
```

### 2.33.6 Отмена валидации

По умолчанию проверяются все валидаторы, присвоенные полю, независимо от того, успешно ли прошла валидация одного из них или нет. Вы можете изменить такое поведение, если укажете валидатору на каком из правил ему следует остановить дальнейшую проверку:

```
<?php

use Phalcon\Validation\Validator\PresenceOf,
    Phalcon\Validation\Validator\Regex;

$validation = new Phalcon\Validation();

$validation
    ->add('telephone', new PresenceOf(array(

```

```
        'message' => 'The telephone is required',
        'cancelOnFail' => true
    ))
->add('telephone', new Regex(array(
    'message' => 'The telephone is required',
    'pattern' => '/\+44 [0-9]+/',
)))
->add('telephone', new StringLength(array(
    'minimumMessage' => 'The telephone is too short',
    'min' => 2
)));
});
```

Первый валидатор имеет свойство 'cancelOnFail' => true, поэтому если валидация не пройдёт эту проверку, то дальнейшие проверки в цепочке не будут выполнены.

Если вы создаёте собственные валидаторы, то можете динамически останавливать их используя свойство 'cancelOnFail':

```
<?php

use Phalcon\Validation\Validator,
    Phalcon\Validation\ValidatorInterface,
    Phalcon\Validation\Message;

class MyValidator extends Validator implements ValidatorInterface
{

    /**
     * Выполняем проверку
     *
     * @param Phalcon\Validation $validator
     * @param string $attribute
     * @return boolean
     */
    public function validate($validator, $attribute)
    {
        // Если имя атрибута 'name' - останавливаем дальнейшие проверки
        if ($attribute == 'name') {
            $validator->setOption('cancelOnFail', true);
        }

        //...
    }
}
```

## 2.34 Формы

Компонент Phalcon\Forms позволяет создавать и управлять формами вашего приложения.

Ниже представлен базовый пример работы с формами:

```
<?php

use Phalcon\Forms\Form,
    Phalcon\Forms\Element\Text,
    Phalcon\Forms\Element>Select;
```

```
$form = new Form();

$form->add(new Text("name"));

$form->add(new Text("telephone"));

$form->add(new Select("telephoneType", array(
    'H' => 'Home',
    'C' => 'Cell'
)));

```

Элементы форм выводятся по указанным при создании именам:

```
<h1>Контакты</h1>

<form method="post">

    <p>
        <label>Имя</label>
        <?php echo $form->render("name") ?>
    </p>

    <p>
        <label>Телефон</label>
        <?php echo $form->render("telephone") ?>
    </p>

    <p>
        <label>Тип телефона</label>
        <?php echo $form->render("telephoneType") ?>
    </p>

    <p>
        <input type="submit" value="Сохранить" />
    </p>

</form>
```

Каждый элемент формы может быть настроен по желанию разработчика. Внутри компонент использует возможности *Phalcon|Tag* для генерации HTML кода каждого документа, вы можете передавать дополнительные html-атрибуты вторым параметром:

```
<p>
    <label>Имя</label>
    <?php echo $form->render("name", array('maxlength' => 30, 'placeholder' => 'Введите своё имя')) ?>
</p>
```

Атрибуты HTML могут быть указаны в параметрах при создании элемента:

```
<?php

$form->add(new Text("name", array(
    'maxlength' => 30,
    'placeholder' => 'Введите своё имя'
)));

```

## 2.34.1 Инициализация

Как уже говорилось ранее, формы могут быть инициализированы вне форм класса путем добавления элементов к нему. Вы можете повторно использовать код или организовать формы собранные из разных файлов:

```
<?php

use Phalcon\Forms\Form,
    Phalcon\Forms\Element\Text,
    Phalcon\Forms\Element>Select;

class ContactForm extends Form
{
    public function initialize()
    {
        $this->add(new Text("name"));

        $this->add(new Text("telephone"));

        $this->add(new Select("telephoneType", TelephoneTypes::find(), array(
            'using' => array('id', 'name')
        )));
    }
}
```

Формы `Phalcon\Forms\Form` наследуются от `Phalcon\DI\Injectable`, предоставляя доступ к службам приложения, если это необходимо:

```
<?php

use Phalcon\Forms\Form,
    Phalcon\Forms\Element\Text,
    Phalcon\Forms\Element\Hidden;

class ContactForm extends Form
{

    /**
     * Этот метод возвращает значение по умолчанию для поля 'csrf'
     */
    public function getCsrf()
    {
        return $this->security->getToken();
    }

    public function initialize()
    {

        // Установка сущности
        $this->setEntity($this);

        // Установка поля 'email'
        $this->add(new Text("email"));

        // Добавление скрытого поля csrf
        $this->add(new Hidden("csrf"));
    }
}
```

При инициализации формы в конструктор передаётся объект пользователя и другие параметры:

```
<?php

use Phalcon\Forms\Form,
    Phalcon\Forms\Element\Text,
    Phalcon\Forms\Element\Hidden;

class UsersForm extends Form
{
    /**
     * Инициализация формы
     *
     * @param Users $user
     * @param array $options
     */
    public function initialize($user, $options)
    {

        if ($options['edit']) {
            $this->add(new Hidden('id'));
        } else {
            $this->add(new Text('id'));
        }

        $this->add(new Text('name'));
    }
}
```

Теперь можно использовать экземпляр формы:

```
<?php

$form = new UsersForm(new Users(), array('edit' => true));
```

## 2.34.2 Валидация

Формы в Phalcon интегрированы с компонентом *валидации* для быстрой проверки введённых данных. Для каждого элемента формы можно устанавливать готовый или настраиваемый валидатор:

```
<?php

use Phalcon\Forms\Element\Text,
    Phalcon\Validation\Validator\PresenceOf,
    Phalcon\Validation\Validator\StringLength;

$name = new Text("name");

$name->addValidator(new PresenceOf(array(
    'message' => 'Поле Name обязательно для заполнения',
)));
$name->addValidator(new StringLength(array(
    'min' => 10,
    'messageMinimum' => 'Значение поля Name слишком короткое',
)));
$form->add($name);
```

Затем вы сможете проверить правильность заполнения формы пользователем:

```
<?php

if (!$form->isValid($_POST)) {
    foreach ($form->getMessages() as $message) {
        echo $message, '<br>';
    }
}
```

Валидаторы выполняются в порядке регистрации.

По умолчанию, сообщения, генерируемые всеми элементами формы, объединены, чтобы их можно было собрать одним проходом foreach, вы можете изменить это поведение, чтобы получить сообщения, разделенные по типам:

```
<?php

foreach ($form->getMessages(false) as $attribute => $messages) {
    echo 'Сообщение создано ', $attribute, ':', "\n";
    foreach ($messages as $message) {
        echo $message, '<br>';
    }
}
```

Так же можно получить сообщения конкретного элемента:

```
<?php

foreach ($form->getMessagesFor('name') as $message) {
    echo $message, '<br>';
}
```

### 2.34.3 Фильтрация

Форма может фильтровать данные до валидации, вы можете установить фильтры в каждом из элементов:

### 2.34.4 Настройка пользовательских параметров

### 2.34.5 Формы и сущности

Модели или коллекции являются такими сущностями, которые можно без проблем связать с формами, их значения в таком случае будут использоваться по умолчанию для соответствующих по именам значений элементов форм. Всё это делается очень легко:

```
<?php

$robot = Robots::findFirst();

$form = new Form($robot);

$form->add(new Text("name"));

$form->add(new Text("year"));
```

При отображении формы, если нет значений по умолчанию для элементов, будут использованы значения из сущностей:

```
<?php echo $form->render('name') ?>
```

Проверить введённые пользователем значения в форму можно следующим образом:

```
<?php
```

```
$form->bind($_POST, $robot);

// Проверка правильности введённых данных формы
if ($form->isValid()) {

    // Сохранение сущности
    $robot->save();
}
```

Установка обычного класса в качестве сущности тоже возможна:

```
<?php
```

```
class Preferences
{

    public $timezone = 'Europe/Amsterdam';

    public $receiveEmails = 'No';

}
```

Использование данного класса в виде сущности позволяет форме брать из него значения по умолчанию:

```
<?php
```

```
$form = new Form(new Preferences());

$form->add(new Select("timezone", array(
    'America/New_York' => 'New York',
    'Europe/Amsterdam' => 'Amsterdam',
    'America/Sao_Paulo' => 'Sao Paulo',
    'Asia/Tokio' => 'Tokio',
)));

$form->add(new Select("receiveEmails", array(
    'Yes' => 'Yes, please!',
    'No' => 'No, thanks'
)));



```

Сущности могут содержать геттеры, приоритет которых выше, чем у публичных свойств. Эти методы дают вам больше свободы для работы со значениями:

```
<?php
```

```
class Preferences
{

    public $timezone;

    public $receiveEmails;
```

```
public function getTimezone()
{
    return 'Europe/Amsterdam';
}

public function getReceiveEmails()
{
    return 'No';
}

}
```

## 2.34.6 Элементы форм

Phalcon предоставляет набор элементов для использования в ваших формах:

Название	Описание	Пример использования
Text	Генерирует элемент INPUT[type=text]	<a href="#">Пример</a>
Password	Генерирует элемент INPUT[type=password]	<a href="#">Пример</a>
Select	Генерирует элемент раскрывающегося списка SELECT	<a href="#">Пример</a>
Check	Генерирует элемент INPUT[type=checkbox]	<a href="#">Пример</a>
Textarea	Генерирует элемент TEXTAREA	<a href="#">Пример</a>
Hidden	Генерирует элемент INPUT[type=hidden]	<a href="#">Пример</a>
File	Генерирует элемент INPUT[type=file]	<a href="#">Пример</a>
Date	Генерирует элемент INPUT[type=date]	<a href="#">Пример</a>
Numeric	Генерирует элемент INPUT[type=number]	<a href="#">Пример</a>
Submit	Генерирует элемент INPUT[type=submit]	<a href="#">Пример</a>

## 2.34.7 Дополнительные условия

Когда формы реализованы в виде классов, в них могут быть определены функции обратного вызова: beforeValidation и afterValidation. Данные методы позволяют осуществлять проверки до и после валидации соответственно:

```
<?php

class ContactForm extends Phalcon\Mvc\Form
{
    public function beforeValidation()
    {

    }
}
```

## 2.34.8 Отрисовка форм

Вы можете гибко отрисовывать формы. Данный пример показывает, как отрисовать каждый элемент, используя стандартную процедуру:

```
<?php

<form method="post">
    <?php
```

```
// Проходим через форму
foreach ($form as $element) {

    // Собираем все сгенерированные сообщения для текущего элемента
    $messages = $form->getMessagesFor($element->getName());

    if (count($messages)) {
        // Выводим каждый элемент
        echo '<div class="messages">';
        foreach ($messages as $message) {
            echo $message;
        }
        echo '</div>';
    }

    echo '<p>';
    echo '<label for=""', $element->getName(), '">', $element->getLabel(), '</label>';
    echo $element;
    echo '</p>';

}
?>
<input type="submit" value="Send"/>
</form>
```

Или повторно использовать логику в классе формы:

```
<?php

class ContactForm extends Phalcon\Forms\Form
{
    public function initialize()
    {
        //...
    }

    public function renderDecorated($name)
    {
        $element = $this->get($name);

        // Собираем все сгенерированные сообщения для текущего элемента
        $messages = $this->getMessagesFor($element->getName());

        if (count($messages)) {
            // Выводим каждый элемент
            echo '<div class="messages">';
            foreach ($messages as $message) {
                echo $this->flash->error($message);
            }
            echo '</div>';
        }

        echo '<p>';
        echo '<label for=""', $element->getName(), '">', $element->getLabel(), '</label>';
        echo $element;
        echo '</p>';
    }
}
```

В представлении:

```
<?php  
  
echo $element->renderDecorated('name');  
  
echo $element->renderDecorated('telephone');
```

### 2.34.9 Создание элементов форм

В дополнение к элементам форм, которые предоставляет Phalcon, вы можете создавать свои собственные элементы:

```
<?php  
  
use Phalcon\Forms\Element;  
  
class MyElement extends Element  
{  
    public function render($attributes=null)  
    {  
        $html = //... немного html-кода  
        return $html;  
    }  
}
```

### 2.34.10 Менеджер форм

Этот компонент предоставляет доступ к менеджеру форм, который может быть использован разработчиком для регистрации форм и доступа к ним через локатор сервисов:

```
<?php  
  
$di['forms'] = function() {  
    return new Phalcon\Forms\Manager();  
}
```

Формы добавляются к менеджеру форм и в дальнейшем могут быть доступны через уникальное имя:

```
<?php  
  
$this->forms->set('login', new LoginForm());
```

С помощью уникального имени формы могут быть доступны в любой части приложения:

```
<?php  
  
echo $this->forms->get('login')->render();
```

### 2.34.11 Внешние источники

- Vökurö, простое приложение, которое использует конструктор форм для создания форм в приложении, [[Github](#)]

## 2.35 Чтение конфигурации

*Phalcon\|Config* - это компонент для чтения конфигурации в разных форматах (используя адаптеры), и преобразования её в PHP-объекты для использования в приложении.

### 2.35.1 Адаптеры файлов

Доступные адаптеры:

Тип файла	Описание
Ini	Использует INI-файлы для хранения конфигурации. Использует PHP-функцию parse_ini_file.
Array	Использует многомерные массивы PHP для конфигурации. Этот адаптер максимально производителен.

### 2.35.2 Нативные массивы

Следующий пример показывает, как конвертировать нативный массив в объект Phalcon\Config. Адаптер для нативных массивов более производителен, так как файлы не разбираются при обращении.

```
<?php
```

```
$settings = array(
    "database" => array(
        "adapter"  => "Mysql",
        "host"      => "localhost",
        "username"  => "scott",
        "password"  => "cheetah",
        "dbname"    => "test_db",
    ),
    "app" => array(
        "controllersDir" => "../app/controllers/",
        "modelsDir"       => "../app/models/",
        "viewsDir"        => "../app/views/",
    ),
    "mysetting" => "the-value"
);

$config = new \Phalcon\Config($settings);

echo $config->app->controllersDir, "\n";
echo $config->database->username, "\n";
echo $config->mysetting, "\n";
```

Если вы хотите лучшей организации для вашего проекта, можно сохранить массив в другой файл и затем прочитать его.

```
<?php
```

```
require "config/config.php";
$config = new \Phalcon\Config($settings);
```

### 2.35.3 Чтение INI-файлов

Ini-файлы являются довольно распространённым способом хранения конфигурации. Для чтения таких файлов Phalcon\Config использует оптимизированную PHP-функцию parse\_ini\_file. Разделы файла разбиваются в подпункты конфигурации для более лёгкого доступа.

```
[database]
adapter = Mysql
host     = localhost
username = scott
password = cheetah
dbname   = test_db

[phalcon]
controllersDir = "../app/controllers/"
modelsDir      = "../app/models/"
viewsDir       = "../app/views/"

[models]
metadata.adapter = "Memory"
```

Чтение файла можно произвести таким способом:

```
<?php

$config = new \Phalcon\Config\Adapter\Ini("path/config.ini");

echo $config->phalcon->controllersDir, "\n";
echo $config->database->username, "\n";
echo $config->models->metadata->adapter, "\n";
```

### 2.35.4 Объединение конфигураций

Phalcon\Config позволяет объединить объекты конфигурации друг в друга рекурсивно:

```
<?php

$config = new \Phalcon\Config(array(
    'database' => array(
        'host' => 'localhost',
        'dbname' => 'test_db'
    ),
    'debug' => 1
));

$config2 = new \Phalcon\Config(array(
    'database' => array(
        'username' => 'scott',
        'password' => 'secret',
    )
));

$config->merge($config2);

print_r($config);
```

Код выше выдаёт такой результат:

```
Phalcon\Config\Object
(
    [database] => Phalcon\Config\Object
    (
        [host] => localhost
        [dbname] => test_db
        [username] => scott
        [password] => secret
    )
    [debug] => 1
)
```

Существует еще несколько типов адаптеров конфигурации, их можно получить в “Инкубаторе” - Phalcon Incubator

## 2.36 Постраничная навигация (Paginators)

Разделение данных на страницы бывает актуально при необходимости вывести большой объём данных поэтапно. Компонент Phalcon\Paginator предлагает простой и удобный способ для этого случая.

### 2.36.1 Адаптеры данных

Компонент имеет встроенные адаптеры для разделения данных на страницы:

Адаптер	Описание
Нативные массивы (NativeArray)	Использует стандартные PHP-массивы
Модели (Model)	Использует объекты типа Phalcon\Mvc\Model\Resultset
QueryBuilder	Использует объект Phalcon\Mvc\Model\Query\Builder

### 2.36.2 Пример

В примере, приведенном ниже, пагинатор будет использовать в качестве источника результат запроса данных модели, и будет выводить данные по 10 записей на странице:

```
<?php

// Текущая страница
// В контроллерах можно использовать:
// $this->request->getQuery('page', 'int'); // GET
// $this->request->getPost('page', 'int'); // POST
$currentPage = (int) $_GET['page'];

// Набор данных для разбики на страницы
$robots = Robots::find();

// Создаём пагинатор, отображаются 10 элементов на странице, начиная с текущей - $currentPage
$paginator = new \Phalcon\Paginator\Adapter\Model(
    array(
        "data" => $robots,
        "limit"=> 10,
        "page" => $currentPage
    )
);
```

```
// Получение результатов работы плагинатора
$page = $paginator->getPaginate();
```

Переменная `$currentPage` указывает то, какая страница сейчас отображается. Метод `$paginator->getPaginate()` возвращает содержащие данные разбивки объект `$page`. Он может быть использован для вывода данные с разделением на страницы:

```
<table>
    <tr>
        <th>Id</th>
        <th>Имя</th>
        <th>Тип</th>
    </tr>
    <?php foreach ($page->items as $item) { ?>
    <tr>
        <td><?php echo $item->id; ?></td>
        <td><?php echo $item->name; ?></td>
        <td><?php echo $item->type; ?></td>
    </tr>
    <?php } ?>
</table>
```

Объект `$page` также содержит данные для навигации:

```
<a href="/robots/search">Первая</a>
<a href="/robots/search?page=<?= $page->before; ?>">Предыдущая</a>
<a href="/robots/search?page=<?= $page->next; ?>">Следующая</a>
<a href="/robots/search?page=<?= $page->last; ?>">Последняя</a>

<?php echo "Вы на странице ", $page->current, " из ", $page->total_pages; ?>
```

### 2.36.3 Использование адаптера

Пример источника данных, который должен быть использован для каждого адаптера:

```
<?php

// Передача данных модели
$paginator = new \Phalcon\Paginator\Adapter\Model(
    array(
        "data"  => Products::find(),
        "limit" => 10,
        "page"  => $currentPage
    )
);

// Передача данных из массива
$paginator = new \Phalcon\Paginator\Adapter\NativeArray(
    array(
        "data"  => array(
            array('id' => 1, 'name' => 'Artichoke'),
            array('id' => 2, 'name' => 'Carrots'),
            array('id' => 3, 'name' => 'Beet'),
            array('id' => 4, 'name' => 'Lettuce'),
            array('id' => 5, 'name' => '')
        ),
        "limit" => 2,
```

```

        "page" => $currentPage
    )
);

// Передача данных querybuilder

$builder = $this->modelsManager->createBuilder()
    ->columns('id, name')
    ->from('Robots')
    ->orderBy('name');

$paginator = new Phalcon\Paginator\Adapter\QueryBuilder(array(
    "builder" => $builder,
    "limit"=> 20,
    "page" => 1
));

```

#### 2.36.4 Атрибуты страниц

Объект `$page` содержит следующие атрибуты:

Атрибут	Описание
items	Набор записей для вывода на текущей странице
current	Текущая страница
before	Номер предыдущей страницы
next	Номер следующей страницы
last	Номер последней страницы
total_pages	Общее число страниц
total_items	Число записей в источнике

#### 2.36.5 Реализация собственных адаптеров

Для создания адаптера необходимо реализовать интерфейс `Phalcon\Paginator\AdapterInterface` или расширить существующий:

```

<?php

class MyPaginator implements Phalcon\Paginator\AdapterInterface
{

    /**
     * Конструктор адаптера
     *
     * @param array $config
     */
    public function __construct($config);

    /**
     * Установка текущей страницы
     *
     * @param int $page
     */
    public function setCurrentPage($page);

    /**

```

```
* Возвращает срез данных для вывода
*
* @return stdClass
*/
public function getPaginate();

}
```

## 2.37 Улучшение производительности с помощью Кэширования

Phalcon предоставляет класс *Phalcon\Cache*, дающий быстрый доступ к часто используемым или уже сгенерированным данным. *Phalcon\Cache* написан на языке C, поэтому он предоставляет высокую производительность и пониженный расход ресурсов. Этот класс использует два компонента: frontend и backend. Frontend компонент является входным источником или интерфейсом, в то время как backend предоставляет опции хранения данных.

### 2.37.1 Где применять кэширование?

Несмотря на то, что этот компонент очень быстрый, его использование в случаях, где он не нужен, может привести к потери производительности. Мы рекомендуем проверить эти ситуации, прежде, чем использовать кэширование:

- Вы делаете сложные расчеты, которые каждый раз возвращают один и тот же результат (или результат редко изменяется)
- Вы используете много помощников (helper), и результат генерации почти всегда одинаковый
- Получение данных из БД, которые редко меняются

*Примечание* Даже, после создания кэширования вы должны проверить скорость выполнения. Это можно легко проверить, особенно используя Memcache или Arc, с помощью соответствующих инструментов, предоставляемыми этими приложениями.

### 2.37.2 Поведение системы кэширования

Процесс кэширования делится на две части:

- **Frontend:** Эта часть отвечает за проверку времени жизни ключа и выполняет дополнительные преобразования над данными, до операции сохранения или извлечения их из backend
- **Backend:** Эта часть отвечает за коммуникацию, запись/чтение данных по запросу frontend

### 2.37.3 Кэширование выходных фрагментов

Выходные фрагменты - это части HTML или текста, которые кэшируются “как есть” и возвращаются “как есть”. Выходные данные автоматически захватываются из `ob_*` функции или из выходного потока PHP и сохраняются в кэш. Следующий пример демонстрирует такое использование. Он получает сгенерированные выходные данные и сохраняет их в файл. Кэш обновляется каждые 172800 секунд (2 дня).

Реализация этого механизма позволяет нам повысить производительность за счет исключения работы помощника `Phalcon\Tag::linkTo`, который вызывается каждый раз в этом участке кода.

```
<?php

// Создание frontend для выходных данных. Кэшируем файлы на 2 дня
$frontCache = new Phalcon\Cache\Frontend\Output(array(
    "lifetime" => 172800
));

// Создаем компонент, который будем кэшировать из "Выходных данных" в "Файл"
// Устанавливаем папку для кэшируемых файлов - важно указать символ "/" в конце пути
$cache = new Phalcon\Cache\Backend\File($frontCache, array(
    "cacheDir" => "../app/cache/"
));

// Получить/Создать кэшируемый файл ../app/cache/my-cache.html
$content = $cache->start("my-cache.html");

// Если $content является значением NULL, значит данных в кэше нет и их надо сгенерировать
if ($content === null) {

    // Выводим дату и время
    echo date("r");

    // Генерируем ссылку на "регистрацию"
    echo Phalcon\Tag::linkTo(
        array(
            "user/signup",
            "Sign Up",
            "class" => "signup-button"
        )
    );

    // Сохраняем вывод в кэш
    $cache->save();
}

} else {

    // Выводим кэшируемые данные
    echo $content;
}
}
```

*Примечание* В этом примере наш код остается таким же и выводит те же данные пользователю. Наш компонент кэширования незаметно перехватывает вывод и сохраняет его в кэшируемый файл (когда кэш сгенерирован) или он отправляет уже готовые данные обратно к пользователю, а это естественно позволяет экономить на выполнении операций.

#### 2.37.4 Кэширование произвольных данных

Кэширование различных данных, не менее важно для вашего приложения. Кэширование может уменьшить нагрузку базы данных за счет повторного использования сгенерированных данных (но не обновленных), что и увеличивает скорость выполнения вашего приложения.

#### Пример файлового Backend

Существует файловый адаптер кэширования. Единственным параметром для него является место, где будут храниться закэшированные файлы. Этот параметр называется "cacheDir", в него передается путь к месту хранения, *важно* добавлять в конце символ "/".

```
<?php

// Кэшируем данные на 2 дня
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 172800
));

// Создаем компонент, который будем кэшировать из "Выходных данных" в "Файл"
// Устанавливаем папку для кэшируемых файлов - важно сохранить символ "/" в конце пути
$cache = new Phalcon\Cache\Backend\File($frontCache, array(
    "cacheDir" => "../app/cache/"
));

// Пробуем получить закэшированные записи
$cacheKey = 'robots_order_id.cache';
$robots = $cache->get($cacheKey);
if ($robots === null) {

    // $robots может иметь значение NULL из-за того, что истекла годность хранения или данных просто не существует
    // Получим данные из БД
    $robots = Robots::find(array("order" => "id"));

    // Сохраняем их в кэше
    $cache->save($cacheKey, $robots);
}

// Используем $robots :)
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

// Используем $robot :)
```

## Пример использования Memcached в качестве Backend

Для этого нам достаточно немного изменить вышеизложенный пример. В частности изменится конфигурация.

```
<?php

// Кэшируем данные на 1 час
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 3600
));

// Создаем компонент, который будет кэшировать данные в Memcache
// Настройки подключения к Memcache
$cache = new Phalcon\Cache\Backend\Memcache($frontCache, array(
    "host" => "localhost",
    "port" => "11211"
));

// Пробуем получить закэшированные записи
$cacheKey = 'robots_order_id.cache';
$robots = $cache->get($cacheKey);
if ($robots === null) {

    // $robots может иметь значение NULL из-за того, что истекла годность хранения или данных просто не существует
    // Получим данные из БД
```

```

$robots = Robots::find(array("order" => "id"));

// Сохраняем их в кэше
$cache->save($cacheKey, $robots);
}

// Используем $robots :)
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

```

## 2.37.5 Запрос данных из кэша

Все элементы добавляемые в кэш идентифицируются по ключам. В случае с файловым backend ключом является название файла. Для получения данных из кэша нам необходимо выполнить запрос к кэшу с указанием уникального ключа. Если ключа не существует, метод вернет значение NULL.

```

<?php

// Получаем продукты по ключу "myProducts"
$products = $cache->get("myProducts");

```

Для того чтобы узнать какие ключи сейчас хранятся можно выполнить метод queryKeys:

```

<?php

// Получаем все ключи, которые хранятся в кэше
$keys = $cache->queryKeys();
foreach ($keys as $key) {
    $data = $cache->get($key);
    echo "Key=", $key, " Data=", $data;
}

// Получаем все ключи, которые начинаются с префикса "my-prefix"
$keys = $cache->queryKeys("my-prefix");

```

## 2.37.6 Удаление данных из кэша

Могут возникнуть ситуации, когда вам необходимо удалить данные из кэша. Единственным требованием для этого является знание необходимого ключа по которому хранятся данные.

```

<?php

// Удаляем элемент по определенному ключу
$cache->delete("someKey");

// Удаляем все из кэша
$keys = $cache->queryKeys();
foreach ($keys as $key) {
    $cache->delete($key);
}

```

## 2.37.7 Проверяем наличие кэша

Существует возможность проверить наличие данных в кэше.

```
<?php

if ($cache->exists("someKey")) {
    echo $cache->get("someKey");
}
else {
    echo "Данных в кэше не существует!";
}
```

## 2.37.8 Время жизни

“Время жизни” (lifetime) - это время, исчисляемое в секундах, которое означает, сколько будут храниться данные в backend кэше. По умолчанию все данные получают “время жизни”, которое было указано при создании frontend компонента. Вы можете указать другое значение при сохранении или получении данных из кэша:

Задаем время жизни при получении:

```
<?php

$cacheKey = 'my.cache';

// Получаем кэш и задаем время жизни
$robots = $cache->get($cacheKey, 3600);
if ($robots === null) {

    $robots = "some robots";

    // Сохраняем в кэше
    $cache->save($cacheKey, $robots);
}
```

Задаем время жизни при сохранении:

```
<?php

$cacheKey = 'my.cache';

$robots = $cache->get($cacheKey);
if ($robots === null) {

    $robots = "some robots";

    // Задаем время жизни, сохраняя данные
    $cache->save($cacheKey, $robots, 3600);
}
```

Существуют некоторые различия в поведении backend компонентов. Например, файловый адаптер требует установку времени жизни при получении, в то время как APC при сохранении.

Во избежание конфликтов можно использовать такую хитрость:

```
<?php
```

```

$lifetime = 3600;
$cacheKey = 'my.cache';

$robots = $cache->get($cacheKey, $lifetime);
if ($robots === null) {

    $robots = "some robots";

    $cache->save($cacheKey, $robots, $lifetime);
}

```

## 2.37.9 Многоуровневое кэширование

Эта возможность компонента кэширования позволяет разработчику осуществлять кэш в несколько уровней. Возможность будет полезна при сохранении кэша в нескольких местах (системах кэширования) с разным временем жизни, и последующим поочерёдным чтением из них начиная с самого быстрого (в порядке регистрации) и заканчивая самым медленным, пока срок жизни во всех них не истечет.

```

<?php

$ultraFastFrontend = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 3600
));

$fastFrontend = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 86400
));

$slowFrontend = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 604800
));

// Backends от самого быстрого до самого медленного
$cache = new \Phalcon\Cache\Multiple(array(
    new Phalcon\Cache\Backend\Apc($ultraFastFrontend, array(
        "prefix" => 'cache',
    )),
    new Phalcon\Cache\Backend\Memcache($fastFrontend, array(
        "prefix" => 'cache',
        "host" => "localhost",
        "port" => "11211"
    )),
    new Phalcon\Cache\Backend\File($slowFrontend, array(
        "prefix" => 'cache',
        "cacheDir" => "../app/cache/"
    ))
));

// Сохраняем, сохраняется сразу во все адаптеры кэширования
$cache->save('my-key', $data);

```

## 2.37.10 Frontend Адаптеры

Доступные адаптеры приведены в таблице:

Адаптер	Описание	Пример
Output	Считывает данные из стандартного PHP вывода	<a href="#">Phalcon\Cache\Frontend\Output</a>
Data	Используется для кэширования любых данных в PHP (big arrays, objects, text, и т.д.). Прежде чем сохранить данные, адаптер сериализирует их.	<a href="#">Phalcon\Cache\Frontend\Data</a>
Base64	Используется для кэширования бинарных данных. Данные сериализируются с использованием base64_encode.	<a href="#">Phalcon\Cache\Frontend\Base64</a>
Json	Данные перед кешированием сериализуются в JSON. Можно использовать для обмена данными с другими фреймворками.	<a href="#">Phalcon\Cache\Frontend\Json</a>
IgBinary	Он используется для кэширования любых данных PHP (большие массивы, объекты, тексты и т.д.). Данные сериализуются с помощью IgBinary перед сохранением в бэкэнд.	<a href="#">Phalcon\Cache\Frontend\Igbinary</a>
None	Используется для кэширования любых типов данных без сериализации.	<a href="#">Phalcon\Cache\Frontend\None</a>

### Реализация собственных Frontend адаптеров

Для создания адаптера необходимо реализовать интерфейс [Phalcon\Cache\FrontendInterface](#).

### 2.37.11 Backend Адаптеры

Доступные адаптеры приведены в таблице:

Адаптер	Описание	Информация	Необходимо дополнение	Пример
File	Сохраняет данные в локальный текстовый файл			<a href="#">Phalcon\Cache\Backend\File</a>
Memcached	Сохраняет данные на memcached сервере	Memcached	memcache	<a href="#">Phalcon\Cache\Backend\Memcache</a>
APC	Сохраняет данные в Alternative PHP Cache (APC)	APC	APC extension	<a href="#">Phalcon\Cache\Backend\Apc</a>
Mongo	Сохраняет данные в Mongo БД	MongoDb	Mongo	<a href="#">Phalcon\Cache\Backend\Mongo</a>
XCache	Сохраняет данные в XCache	XCache	xcache extension	<a href="#">Phalcon\Cache\Backend\Xcache</a>

### Реализация собственных Backend адаптеров

Для создания адаптера необходимо реализовать интерфейс [Phalcon\Cache\BackendInterface](#).

#### Опции файлового Backend

Данные будут сохранены в файлы на локальном сервере. Доступные опции:

Опция	Описание
prefix	Префикс, который будет автоматически добавляться в кэш ключей
cacheDir	Папка с правами на запись, в которую будут сохраняться кэшируемые файлы

#### Опции Memcached Backend

Данные будут сохранены на memcached сервере. Доступные опции:

Опция	Описание
prefix	Префикс, который будет автоматически добавляться в кэш ключей
host	Адрес сервера memcached
port	Порт сервера memcached
persistent	Использовать постоянное соединение с memcached?

## Опции APC Backend

Данные будут сохранены в Alternative PHP Cache (APC). Доступна лишь одна опция:

Опция	Описание
prefix	Префикс, который будет автоматически добавляться в кэш ключей

## Опции Mongo Backend

Данные будут сохранены на MongoDB сервере. Доступные опции:

Опция	Описание
prefix	Префикс, который будет автоматически добавляться в кэш ключей
server	Строка подключения к MongoDB
db	Название базы данных
collection	Коллекция в базе данных

## Опции XCache Backend

Данные будут сохранены в кэше XCache (XCache). Доступна лишь одна опция:

Опция	Описание
prefix	Префикс, который будет автоматически добавляться в кэш ключей

Существует еще несколько типов адаптеров, их можно получить в “Инкубаторе” - Phalcon Incubator

## 2.38 Безопасность

Этот компонент помогает разработчику в общих задачах обеспечения безопасности, таких как хеширование паролей и защита от атак вида Cross-Site Request Forgery (CSRF).

### 2.38.1 Хеширование паролей

Хранение паролей в открытом виде является плохой практикой. Любой, кто имеет доступ к базе данных, мгновенно получит доступ ко всем пользовательским аккаунтам и, таким образом, получает возможность производить неавторизованные действия. Для противостояния этому, многие приложения используют знакомые методы одностороннего хеширования вроде “md5” и “sha1”. Однако аппаратное обеспечение развивается с каждым днем, становится быстрее, и эти алгоритмы становятся уязвимы к атакам методом перебора. Данные атаки также известны как “радужные таблицы” (rainbow tables).

Для решения этой проблемы, мы можем использовать такие алгоритмы хеширования, как `bcrypt`. Почему `bcrypt`? Благодаря алгоритму установки ключа “Eksblowfish” мы можем сделать шифрование пароля настолько “медленным”, насколько мы этого захотим. Медленные алгоритмы делают процесс вычисления настоящего пароля, скрытого за хешем, крайне сложным, если не невозможным. Это защитит вас на долгое время от возможных атак с использованием радужных таблиц.

Этот компонент дает вам возможность простым способом использовать данный алгоритм:

```
<?php

use Phalcon\Mvc\Controller;

class UsersController extends Controller
{

    public function registerAction()
    {

        $user = new Users();

        $login = $this->request->getPost('login');
        $password = $this->request->getPost('password');

        $user->login = $login;

        //Сохраняем пароль хешированным
        $user->password = $this->security->hash($password);

        $user->save();
    }

}
```

Мы сохранили пароль хешированным с коэффициентом хеширования по-умолчанию. Более высокий коэффициент хеширования сделает пароль менее уязвимым, так как его шифрование будет медленным. Мы можем проверить правильность пароля следующим способом:

```
<?php

use Phalcon\Mvc\Controller;

class SessionController extends Controller
{

    public function loginAction()
    {

        $login = $this->request->getPost('login');
        $password = $this->request->getPost('password');

        $user = Users::findFirst(array(
            "login = ?0",
            "bind" => array($login)
        ));
        if ($user) {
            if ($this->security->checkHash($password, $user->password)) {
                //Пароль верный
            }
        }

        //неудачная проверка
    }

}
```

Соль генерируется с использованием псевдослучайных байтов функции PHP `openssl_random_pseudo_bytes`, поэтому необходимо, чтобы расширение `openssl` было загружено.

## 2.38.2 Защита от Cross-Site Request Forgery (CSRF)

Это один из других видов атак на веб-сайты и приложения. Формы, созданные для выполнения таких задач, как регистрация или добавление комментариев, уязвимы для этих атак.

Основной идеей является предотвращение отправления значений формы куда-либо вне нашего приложения. Чтобы это сделать, мы генерируем токен (`nonce`) для каждой формы, добавляем этот токен в сессию, а после, как только форма возвращает данные нашему приложению, проверяем токен, сравнивая присланный формой токен с его сохраненным значением в сессии:

```
<?php echo Tag::form('session/login') ?>

<!-- поля логина и пароля ... -->

<input type="hidden" name="<?php echo $this->security->getTokenKey() ?>" value="<?php echo $this->security->getToken() ?>"/>

</form>
```

После этого, в действии контроллера вы можете проверить CSRF-токен на правильность:

```
<?php

use Phalcon\Mvc\Controller;

class SessionController extends Controller
{

    public function loginAction()
    {
        if ($this->request->isPost()) {
            if ($this->security->checkToken()) {
                //Токен верный
            }
        }
    }
}
```

Также рекомендуется добавление каптчи (`captcha`) в форму, чтобы полностью избежать рисков от этого типа атак.

## 2.38.3 Настройка компонента

Компонент автоматически регистрируется в контейнере сервисов под названием ‘`security`’, вы можете его перерегистрировать для настройки параметров:

```
<?php

$di->set('security', function(){

    $security = new Phalcon\Security();

    //Устанавливаем фактор хеширования в 12 раундов
```

```
$security->setWorkFactor(12);

    return $security,
}, true);
```

## 2.38.4 Внешние источники

- Vökuró, пример приложения с использованием Security для избежание CSRF и хешированием паролей [[Github](#)]

## 2.39 Зашифрование и расшифрование

Phalcon предоставляет средства шифрования с помощью компонента *Phalcon\Crypt*. Этот класс предоставляет простые объектно-ориентированные обертки к php библиотеке *mcrypt*.

По умолчанию данный компонент использует надежный алгоритм шифрования AES-256 (rijndael-256-cbc).

### 2.39.1 Базовое использование

Данный компонент разработан так, чтобы быть максимально простым в использовании:

```
<?php

// Создание экземпляра
$crypt = new Phalcon\Crypt();

$key = 'это пароль';
$text = 'Это секретный текст';

$encrypted = $crypt->encrypt($text, $key);

echo $crypt->decrypt($encrypted, $key);
```

Вы можете использовать тот же экземпляр для многократного зашифрования или расшифрования:

```
<?php

// Создание экземпляра
$crypt = new Phalcon\Crypt();

$texts = array(
    'my-key' => 'Это секретный текст',
    'other-key' => 'Это очень секретно'
);

foreach ($texts as $key => $text) {

    // Зашифровываем
    $encrypted = $crypt->encrypt($text, $key);

    // Теперь расшифровываем
    echo $crypt->decrypt($encrypted, $key);
}
```

## 2.39.2 Атрибуты шифрования

Для изменения в поведении шифрования доступны следующие атрибуты:

Атрибут	Описание
Cipher	Один из алгоритмов шифрования поддерживаемый libmcrypt. Посмотреть список Вы можете <a href="#">здесь</a>
Mode	Один из режимов шифрования поддерживаемый libmcrypt (ecb, cbc, cfb, ofb)

Пример:

```
<?php

// Создаем экземпляр
$crypt = new Phalcon\Crypt();

// Используем алгоритм blowfish
$crypt->setCipher('blowfish');

$key = 'это пароль';
$text = 'Это секретный текст';

echo $crypt->encrypt($text, $key);
```

## 2.39.3 Поддержка Base64

Для того, чтобы зашифрованный текст должным образом передать (по электронной почте) или отобразить (в браузере) очень часто применяется кодирование base64.

```
<?php

// Создаем экземпляр
$crypt = new Phalcon\Crypt();

$key = 'это пароль';
$text = 'Это секретный текст';

$encrypt = $crypt->encryptBase64($text, $key);

echo $crypt->decryptBase64($text, $key);
```

## 2.39.4 Настройка сервиса

Чтобы использовать компонент шифрования из любой точки приложения, Вы можете поместить его в контейнер сервисов:

```
<?php

$di->set('crypt', function() {

    $crypt = new Phalcon\Crypt();

    // Устанавливаем глобальный ключ шифрования
    $crypt->setKey('%31.1e$i86e$f!8jz');
```

```
    return $crypt;
}, true);
```

Затем, как пример, Вы можете использовать его в контроллере следующим образом:

```
<?php

use Phalcon\Mvc\Controller;

class SecretsController extends Controller
{

    public function saveAction()
    {
        $secret = new Secrets();

        $text = $this->request->getPost('text');

        $secret->content = $this->crypt->encrypt($text);

        if ($secret->save()) {
            $this->flash->success('Секрет успешно создан!');
        }
    }

}
```

## 2.40 Списки Контроля Доступа (ACL)

*Phalcon\Acl* предоставляет простое и легкое управление контролем доступа и прикрепленными разрешениями. Список контроля доступа (ACL) позволяет приложению управлять доступом к своим областям и основным запрошенным объектам. Необходимо понимать основные методологии ACL, чтобы понимать принцип работы.

И в заключении: ACL состоит из ролей и ресурсов. Ресурсами являются объекты, на которые накладываются определенные разрешения с помощью ACL. Ролями являются объекты, которые запрашивают доступ к ресурсам и получают ответ от ACL: разрешено/запрещено.

### 2.40.1 Создание ACL

Этот компонент изначально сделан так, чтобы работать непосредственно в памяти. Это предоставляет простое использование и скорость в обращении к любому аспекту списка. Конструктор *Phalcon\Acl* принимает в качестве первого параметра адаптер, который будет использоваться для получения информации связанной с контролируемым списком. Пример использования адаптера памяти:

```
<?php $acl = new \Phalcon\Acl\Adapter\Memory();
```

По умолчанию *Phalcon\Acl* предоставляет доступ к действию над ресурсом, которое еще не было определено в ACL. Чтобы увеличить уровень безопасности мы можем указать уровень “запрещено”, как уровень по умолчанию.

```
<?php
```

```
// Указываем "запрещено" по умолчанию для тех объектов, которые не были занесены в список контроля доступа
$acl->setDefaultAction(Phalcon\Acl::DENY);
```

## 2.40.2 Добавление ролей

Ролью является объект, который имеет или не имеет доступа к определенному ресурсу в списке доступа. Для примера, мы определим роли людей из организации. Класс *Phalcon\Acl\Role* позволяет создать роли в более структурированной форме. Давайте добавим несколько ролей в наш недавно созданный список:

```
<?php

// Создаем роли
$roleAdmins = new \Phalcon\Acl\Role("Administrators", "Super-User role");
$roleGuests = new \Phalcon\Acl\Role("Guests");

// Добавляем "Guests" в список ACL
$acl->addRole($roleGuests);

// Добавляем "Designers" без класса Phalcon\Acl\Role
$acl->addRole("Designers");
```

Как вы можете видеть, роли определяются непосредственно, без использования экземпляра.

## 2.40.3 Добавление ресурсов

Ресурсами являются объекты, доступ к которым контролируется. Обычно в MVC приложениях ресурсы относятся к контроллерам. Хотя это не является обязательным, класс *Phalcon\Acl\Resource* может быть использован при определении любых ресурсов. Важно добавить связующие действия или операции над ресурсами, чтобы ACL мог понимать, что ему нужно контролировать.

```
<?php

// Определяем ресурс "Customers"
$customersResource = new \Phalcon\Acl\Resource("Customers");

// Добавим ресурс "Customers" с несколькими операциями
$acl->addResource($customersResource, "search");
$acl->addResource($customersResource, array("create", "update"));
```

## 2.40.4 Определение контроля доступа

Теперь у нас есть роли и ресурсы. Настало время указать для ACL, какие разрешения имеют роли при доступе к ресурсам. Данная часть очень важна, особенно принимая во внимание используемый по умолчанию уровень “разрешить” или “запретить”.

```
<?php

// Задаем уровень доступа для ролей на определенный ресурс
$acl->allow("Guests", "Customers", "search");
$acl->allow("Guests", "Customers", "create");
$acl->deny("Guests", "Customers", "update");
```

Метод “allow” определяет, что данная роль имеет доступ к действию над ресурсом. Метод “deny” делает обратное.

## 2.40.5 Запросы к ACL

После того, как список был полностью составлен, мы можем запрашивать проверку на права той или иной роли.

```
<?php
```

```
// Проверяем, имеет ли роль "Guests" доступ к разным операциям по отношению к ресурсу "Customers"
$acl->isAllowed("Guests", "Customers", "edit"); // Возвращает 0
$acl->isAllowed("Guests", "Customers", "search"); // Возвращает 1
$acl->isAllowed("Guests", "Customers", "create"); // Возвращает 1
```

## 2.40.6 Наследование ролей

Вы можете строить сложные структуры ролей используя наследование, которое предоставляет класс `Phalcon\Acl\Role`. Роли могут наследовать доступ других ролей. Чтобы использовать наследование ролей вам необходимо передать в качестве второго параметра другую роль при определении роли.

```
<?php
```

```
// Создаем несколько ролей
$roleAdmins = new \Phalcon\Acl\Role("Administrators", "Super-User role");
$roleGuests = new \Phalcon\Acl\Role("Guests");

// Добавляем роль "Guests"
$acl->addRole($roleGuests);

// Добавляем роль "Administrators" наследуемую от роли "Guests"
$acl->addRole($roleAdmins, $roleGuests);
```

## 2.40.7 Сериализация ACL

Чтобы увеличить производительность, объект `Phalcon\Acl` можно сериализовать для хранения в текстовом формате или в базе данных, и повторно использовать `Phalcon\Acl` без переобъявления всего списка каждый раз. Вы можете сделать это следующим образом:

```
<?php
```

```
// Проверяем существует ли сериализованный файл
if (!file_exists("app/security/acl.data")) {

    $acl = new \Phalcon\Acl\Adapter\Memory();

    //... Определяем роли, ресурсы, доступ и т.д.

    // Сохраняем сериализованный объект в файл
    file_put_contents("app/security/acl.data", serialize($acl));

} else {

    // Восстанавливаем ACL объект из текстового файла
```

```

    $acl = unserialize(file_get_contents("app/security/acl.data"));
}

// Используем ACL
if ($acl->isAllowed("Guests", "Customers", "edit")) {
    echo "Доступ разрешен!";
} else {
    echo "Доступ запрещен :(";
}

```

## 2.40.8 События ACL

*Phalcon\Acl* может отправлять события в *EventsManager*. События срабатывают используя тип “acl”. Некоторые события могут возвращать boolean значение ‘false’, чтобы прервать текущую операцию. Поддерживаются следующие типы событий:

Название события	Когда срабатывает	Может остановить операцию?
beforeCheckAccess	Срабатывает перед проверкой доступа роли/ресурса	Да
afterCheckAccess	Срабатывает после проверки доступа роли/ресурса	Нет

В следующем примере показано, как прикрепить слушателей (listeners) к компоненту:

```

<?php

// Создаем менеджер событий
$eventsManager = new Phalcon\Events\Manager();

// Прикрепляем слушателя (функцию/callback) к типу "acl"
$eventsManager->attach("acl", function($event, $acl) {
    if ($event->getType() == 'beforeCheckAccess') {
        echo $acl->getActiveRole(),
            $acl->getActiveResource(),
            $acl->getActiveAccess();
    }
});

$acl = new \Phalcon\Acl\Adapter\Memory();

// Настраиваем $acl
//...

// Присваиваем менеджера событий к компоненту ACL
$acl->setEventManager($eventManagers);

```

## 2.40.9 Реализация собственных адаптеров

Для создания своего адаптера необходимо реализовать интерфейс *Phalcon\Acl\AdapterInterface*, или использовать наследование от существующего адаптера.

## 2.41 Поддержка многоязычности

Компонент Phalcon\Translate поможет в создании многоязычных приложений. Приложения, использующие этот компонент, отображают контент на разных языках, основываясь на выборе пользователя из поддерживаемых приложением.

### 2.41.1 Адаптеры

Этот компонент позволяет использовать адаптеры для чтения, перевода сообщений из различных источников в едином виде.

Адаптер	Описание
NativeArray	Использует PHP массивы для хранения. Это лучший вариант с точки зрения производительности.

### 2.41.2 Использование компонента

Строки переводов хранятся в файлах. Структура этих файлов может отличаться в зависимости от используемого адаптера. Phalcon дает вам свободу выбора в организации правил перевода строк. Типичной структурой может быть:

```
app/messages/en.php
app/messages/es.php
app/messages/fr.php
app/messages/zh.php
```

Каждый файл содержит массив строк с переводами в виде ключ=>значение. Для каждого файла перевода, ключи уникальны. Один и тот же массив используется в разных файлах, ключи в нём остаются прежними, а значения содержат переводы строк, разные для разных языков.

```
<?php

//app/messages/en.php
$messages = array(
    "hi"      => "Hello",
    "bye"     => "Good Bye",
    "hi-name" => "Hello %name%",
    "song"    => "This song is %song%"
);

<?php

//app/messages/ru.php
$messages = array(
    "hi"      => "Здарова",
    "bye"     => "Прощай",
    "hi-name" => "Здарова %name%",
    "song"    => "Композиция %song%"
);
```

Механизм осуществления перевода в приложении тривиален, но зависит от того, как вы хотите реализовать ее. Вы можете использовать автоматическое определение языка из браузера пользователя, или вы можете предоставить выбор языка пользователю.

Простой способ обнаружения языка пользователя - разбор содержимого `$_SERVER['HTTP_ACCEPT_LANGUAGE']`, доступ к нему можно получить непосредственно обратившись в `$this->request->getBestLanguage()` из действия/контроллера:

```
<?php

class UserController extends \Phalcon\Mvc\Controller
{

    protected function _getTranslation()
    {

        // Получение оптимального языка из браузера
        $language = $this->request->getBestLanguage();

        // Проверка существования перевода для полученного языка
        if (file_exists("app/messages/".$language.".php")) {
            require "app/messages/".$language.".php";
        } else {
            // Переключение на язык по умолчанию
            require "app/messages/en.php";
        }

        // Возвращение объекта работы с переводом
        return new \Phalcon\Translate\Adapter\NativeArray(array(
            "content" => $messages
        ));
    }

    public function indexAction()
    {
        $this->view->setVar("name", "Mike");
        $this->view->setVar("t", $this->_getTranslation());
    }
}
```

Метод `_getTranslation` в этом примере доступен для всех действий требующих перевода. Переменная `$t` передается в представление и позволяет непосредственно переводить строки:

```
<!-- welcome -->
<!-- String: hi => 'Hello' -->
<p><?php echo $t->_("hi"), " ", $name; ?></p>
```

The “`_`” function is returning the translated string based on the index passed. Some strings need to incorporate placeholders for calculated data i.e. Hello %name%. These placeholders can be replaced with passed parameters in the “`_`” function. The passed parameters are in the form of a key/value array, where the key matches the placeholder name and the value is the actual data to be replaced:

Функция “`_`” возвращает переведенные строки на основе используемого индекса. В некоторых строках необходимо использовать шаблоны подстановок, например: “Здравствуйте % name%”. Эти подстановки (placeholders) могут быть заменены передаваемыми параметрами в функцию “`_`”. Параметры должны передаваться в виде массива ключ/значение, где ключ соответствует названию подстановки, а значение - фактическим данным для замены:

```
<!-- welcome -->
<!-- String: hi-user => 'Hello %name%' -->
<p><?php echo $t->_("hi-user", array("name" => $name)); ?></p>
```

Существуют так же приложения с многоязычностью основанной на параметрах в URL, например как <http://www.mozilla.org/es-ES/firefox/>. Реализовать такую схему на Phalcon можно используя компонент *Router*.

### 2.41.3 Реализация собственных адаптеров

Для создания адаптера необходимо реализовать интерфейс *Phalcon\Translate\AdapterInterface* или расширить существующий:

```
<?php

class MyTranslateAdapter implements Phalcon\Translate\AdapterInterface
{

    /**
     * Adapter constructor
     *
     * @param array $data
     */
    public function __construct($options);

    /**
     * Возвращает перевод строки по ключу
     *
     * @param string $translateKey
     * @param array $placeholders
     * @return string
     */
    public function _($translateKey, $placeholders=null);

    /**
     * Возвращает перевод, связанный с заданным ключом
     *
     * @param string $index
     * @param array $placeholders
     * @return string
     */
    public function query($index, $placeholders=null);

    /**
     * Проверяет существование перевода ключа во внутреннем массиве
     *
     * @param string $index
     * @return bool
     */
    public function exists($index);

}
```

Больше адаптеров перевода можно найти в [Инкубаторе Phalcon](#)

## 2.42 Универсальный загрузчик классов

Компонент *Phalcon\Loader* позволяет осуществлять автоматическую загрузку классов, основываясь на установленных правилах. Компонент написан на Си и обеспечивает очень низкие накладные расходы

на чтение и интерпретацию PHP-файлов.

Поведение компонента основано на стандартной для PHP возможности [автозагрузки классов](#). Если используемый в коде класс не существует, специальный обработчик будет пытаться найти его и загрузить. *Phalcon\Loader* создан как раз для этой операции. Загрузка только необходимых для работы классов положительно сказывается на производительности приложения, так как в работе участвуют только файлы, непосредственно требуемые для текущей операции. Такая технология называется [отложенная \(ленивая\) инициализация](#).

С помощью этого класса возможна загрузка файлов сторонних проектов и производителей, компонент полностью совместим со [стандартом PSR-0](#).

*Phalcon\Loader* поддерживает 4 варианта правил автозагрузки классов. Вы можете использовать их по отдельности, или комбинировать.

### 2.42.1 Регистрация пространств имён

Если организация вашего кода подразумевает пространства имён, или использованы внешние библиотеки с их использованием, то для регистрации стоит использовать метод `registerNamespaces()`. Метод принимает ассоциативный массив, в котором ключами являются префиксы пространств имен, а их значениями - каталоги в которых эти классы расположены. Разделитель пространства имен (`namespace`) ("\"), будет заменен разделителем директорий (""/"), во время поиска класса загрузчиком. Не забывайте всегда добавлять заключительный слеш в конце пути каталогов:

```
<?php

// Создание загрузчика
$loader = new \Phalcon\Loader();

// Регистрация пространств имён
$loader->registerNamespaces(
    array(
        "Example\Base"      => "vendor/example/base/",
        "Example\Adapter"   => "vendor/example/adapter/",
        "Example"            => "vendor/example/",
    )
);

// Регистрация автозагрузчика
$loader->register();

// Требуемый файл должен располагаться в vendor/example/adapter/Some.php
$some = new Example\Adapter\Some();
```

### 2.42.2 Регистрация префиксов

Эта стратегия похожа на работу с пространствами имён. Для работы используется ассоциативный массив, в котором ключами являются префиксы, а их значениями - каталоги в которых эти классы расположены. Разделитель пространства имен ("\"") и символ подчёркивания "\_" будет заменен разделителем директорий (""/"), во время поиска класса загрузчиком. Не забывайте всегда добавлять заключительный слеш в конце пути каталогов:

```
<?php

// Создание загрузчика
$loader = new \Phalcon\Loader();
```

```
// Регистрация некоторых префиксов
$loader->registerPrefixes(
    array(
        "Example_Base"      => "vendor/example/base/",
        "Example_Adapter"   => "vendor/example/adapter/",
        "Example_"           => "vendor/example/",
    )
);

// Регистрация автозагрузчика
$loader->register();

// Требуемый файл будет искаться в vendor/example/adapter/Some.php
$some = new Example_Adapter_Some();
```

### 2.42.3 Регистрация каталогов

Третий вариант - регистрация каталогов для поиска файлов. Этот вариант не очень рекомендуется с точки зрения производительности, при его использовании Phalcon будет вынужден обрабатывать данные по каждому каталогу и искать в них файл с таким же именем что и название требуемого класса. Важно регистрировать каталоги в правильном порядке, так же не забывайте всегда добавлять заключительный слеш в конце пути:

```
<?php

// Создание загрузчика
$loader = new \Phalcon\Loader();

// Регистрация каталогов
$loader->registerDirs(
    array(
        "library/MyComponent/",
        "library/OtherComponent/Other/",
        "vendor/example/adapters/",
        "vendor/example/"
    )
);

// Регистрация автозагрузчика
$loader->register();

// Требуемый файл будет автоматически подключен из первого каталога в котором он будет найден
// например library/OtherComponent/Other/Some.php
$some = new Some();
```

### 2.42.4 Регистрация классов

Последний вариант - регистрация названия класса и пути к нему. Это решение может быть полезно при использовании стратегий, не позволяющих легко получить файл, используя название или путь к классу. Это самый быстрый способ автозагрузки. Но при разрастании приложения, число файлов так же будет расти, увеличивая список автозагрузки. Разрастание списка снижает эффективность и не рекомендуется по вопросам производительности.

```
<?php

// Создание загрузчика
$loader = new \Phalcon\Loader();

// Регистрация классов
$loader->registerClasses(
    array(
        "Some"          => "library/OtherComponent/Other/Some.php",
        "Example\Base" => "vendor/example/adapters/Example/BaseClass.php",
    )
);

// Регистрация автозагрузчика
$loader->register();

// Искомый класс будет искаться на соответствующее зарегистрированное значение массива
// например library/OtherComponent/Other/Some.php
$some = new Some();
```

## 2.42.5 Дополнительные расширения файлов

Автозагрузка с использованием префиксов, пространств имён и регистрации каталогов автоматически добавляет расширение “php” во время поиска файлов. Если у вас используются дополнительные расширения, их можно указать с помощью метода “setExtensions”. Файлы при этом будут проверять в порядке регистрации расширений:

```
<?php

// Создание загрузчика
$loader = new \Phalcon\Loader();

// Установка расширений файлов для поиска классов
$loader->setExtensions(array("php", "inc", "phb"));
```

## 2.42.6 Изменение текущей стратегии

Дополнительные данные могут быть добавлены к существующим значениям стратегии следующим образом:

```
<?php

// Регистрация дополнительных каталогов
$loader->registerDirs(
    array(
        "../app/library/",
        "../app/plugins/"
    ),
    true
);
```

Использование “true” в качестве второго параметра позволит добавить новые значения к уже имеющимся.

## 2.42.7 События автозагрузки классов

В следующем примере, EventsManager работает с загрузчиком класса, что позволяет нам получать отладочную информацию о выполнении работы:

```
<?php

$eventsManager = new \Phalcon\Events\Manager();

$loader = new \Phalcon\Loader();

$loader->registerNamespaces(array(
    'Example\Base' => 'vendor/example/base/',
    'Example\Adapter' => 'vendor/example/adapter/',
    'Example' => 'vendor/example/',
));

// Прослушивание всех событий загрузчика
$eventsManager->attach('loader', function($event, $loader) {
    if ($event->getType() == 'beforeCheckPath') {
        echo $loader->getCheckedPath();
    }
});

$loader->setEventsManager($eventsManager);

$loader->register();
```

Некоторые события при возвращении логического “false” могут остановить активную операцию. Список поддерживаемых событий:

Название события	Условия срабатывания
beforeCheckClass	До начала процесса автозагрузки
pathFound	Когда найдено расположение класса
afterCheckClass	После завершения процесса автозагрузки. Событие вызывается, если автозагрузчик не обнаружил

## 2.42.8 Устранение неполадок

Некоторые вещи, которые стоит иметь в виду при использовании универсального автозагрузчика:

- Загрузчик чувствителен к регистру
- Стратегии, основанные на пространствах имён и префиксах, быстрее, чем стратегии на каталогах
- Если доступен APC, он будет использован для запрашиваемого файла (и этот файл будет кэширован)

## 2.43 Логирование

*Phalcon\Logger* является компонентом для обеспечения ведения логов в приложении. Он позволяет вести логирование разных типов с использованием различных адаптеров. Он также предлагает регистрацию транзакций, параметров конфигурации, различных форматов и фильтров. Вы можете использовать *Phalcon\Logger* для логирования всех операций, отладки процессов и отслеживания работы приложения.

### 2.43.1 Адаптеры

Этот компонент позволяет использовать адаптеры для хранения журнала сообщений. Использование адаптеров обеспечивает общий интерфейс для регистрации время переключения интерфейсов при необходимости. Реализованные адаптеры:

Адаптер	Описание	API
File	Логирование в текстовой файл	<i>Phalcon\Logger\Adapter\File</i>
Stream	Логирование в PHP поток	<i>Phalcon\Logger\Adapter\Stream</i>
Syslog	Логирование в системный журнал	<i>Phalcon\Logger\Adapter\Syslog</i>

### 2.43.2 Создание журнала

В приведенном ниже примере показано, как создать журнал и добавить в него запись:

```
<?php
```

```
use Phalcon\Logger\Adapter\File as FileAdapter;

$logger = new FileAdapter("app/logs/test.log");
$logger->log("Это сообщение");
$logger->log("А это уже сообщение об ошибке", \Phalcon\Logger::ERROR);
$logger->error("Это тоже про ошибку");
```

Результат кода:

```
[Tue, 17 Apr 12 22:09:02 -0500] [DEBUG] Это сообщение
[Tue, 17 Apr 12 22:09:02 -0500] [ERROR] А это уже сообщение об ошибке
[Tue, 17 Apr 12 22:09:02 -0500] [ERROR] Это тоже про ошибку
```

### 2.43.3 Транзакции

Запись данных в адаптер т.е. в файл (файловая система) всегда является ‘дорогостоящей’ операцией с точки зрения производительности. Для решения этой задачи, можно использовать транзакции при логировании. Транзакции временно хранят записи в памяти, а затем переносят их соответствующий адаптер (в данном случае в файл).

```
<?php
```

```
use Phalcon\Logger\Adapter\File as FileAdapter;

// Создание логгера
$logger = new FileAdapter("app/logs/test.log");

// Начало транзакции
$logger->begin();

// Добавление записей
$logger->alert("This is an alert");
$logger->error("This is another error");

// Размещение записей в файл
$logger->commit();
```

## 2.43.4 Одновременное логирование нескольких обработчиков

*Phalcon\Logger* позволяет отправку сообщений на несколько обработчиков одним вызовом:

```
<?php
```

```
use Phalcon\Logger,
    Phalcon\Logger\Multiple as MultipleStream,
    Phalcon\Logger\Adapter\File as FileAdapter,
    Phalcon\Logger\Adapter\Stream as StreamAdapter;

$logger = new MultipleStream();

$logger->push(new FileAdapter('test.log'));
$logger->push(new StreamAdapter('php://stdout'));

$logger->log("This is a message");
$logger->log("This is an error", Logger::ERROR);
$logger->error("This is another error");
```

Сообщения отправляются на обработчик в порядке их регистраций.

## 2.43.5 Форматирование сообщений

Данный компонент позволяет использовать ‘formatters’ для форматирования сообщений перед тем как их отправить на бэкенд. Реализованные следующие форматеры:

Адаптер	Описание	API
Line	Оформление записей одной строкой	<i>Phalcon\Logger\Formatter\Line</i>
Json	Подготовка записей для преобразование в JSON	<i>Phalcon\Logger\Formatter\Json</i>
Syslog	Подготовка записи для отправки в системный журнал	<i>Phalcon\Logger\Formatter\Syslog</i>

### Линейный Оформитель

Оформление записей в одну строку. Формат по умолчанию:

```
[%date%][%type%] %message%
```

Вы можете изменить вид сообщений по умолчанию используя `setFormat()`, этот метод позволяет менять формат конечных сообщений, определяя свой собственный. Поддерживаются такие переменные:

Переменные	Описание
%message%	Запись, которая будет внесена
%date%	Дата добавления записи в журнал
%type%	Тип записи заглавными буквами

В приведенном примере показано, как изменить формат сообщений в логе:

```
<?php
```

```
use Phalcon\Logger\Formatter\Line as LineFormatter;

// Установка формата сообщений в логе
formatter = new LineFormatter("%date% - %message%");
$logger->setFormatter(formatter);
```

## Реализация собственного оформителя

Для создания оформителя необходимо реализовать интерфейс `Phalcon\Logger\FormatterInterface` или расширить существующий.

### 2.43.6 Адаптеры

В Phalcon есть несколько реализованных адаптеров логирования, примеры ниже показывают, как их можно использовать:

#### Stream Logger

Записывает сообщения в зарегистрированные потоки PHP. Поддерживаемые протоколы перечислены [здесь](#):

```
<?php

use Phalcon\Logger\Adapter\Stream as StreamAdapter;

// Открывает поток с использованием zlib компрессии
$logger = new StreamAdapter("compress.zlib://week.log.gz");

// Пишет сообщения в stderr
$logger = new StreamAdapter("php://stderr");
```

#### File Logger

Этот регистратор использует обычные файлы для ведения логов всех типов. По умолчанию все файлы регистратор открыл в режиме добавления записей, размещая новую запись в конце файла. Если файл не существует, регистратор попытается его создать. Вы можете изменить этот режим, передавая дополнительную опцию в конструктор:

```
<?php

use Phalcon\Logger\Adapter\File as FileAdapter;

// Создание регистратора с поддержкой записи
$logger = new FileAdapter("app/logs/test.log", array(
    'mode' => 'w',
));
```

#### Syslog Logger

Этот регистратор отправляет сообщения в системный журнал. Работа такого журнала может варьироваться от одной операционной системы к другой.

```
<?php

use Phalcon\Logger\Adapter\Syslog as SyslogAdapter;

// Основное использование
$logger = new SyslogAdapter(null);
```

```
// Установка ident/mode/facility
$logger = new SyslogAdapter("ident-name", array(
    'option' => LOG_NDELAY,
    'facility' => LOG_MAIL
));
```

## Реализация собственных адаптеров

Для создания адаптера необходимо реализовать интерфейс *Phalcon\Logger\AdapterInterface* или расширить существующий адаптер.

## 2.44 Парсер аннотаций

Это первый прецедент для мира PHP, когда компонент парсера аннотаций написан на С. Phalcon\Annotations - компонент общего назначения, обеспечивающий простоту анализа и кэширования аннотаций для PHP классов используемых в приложениях.

Аннотации читаются из блоков комментариев docblock в классах, его методах и свойствах. Аннотации могут быть помещены в любое место блока документации docblock.

```
<?php

/**
 * Это описание класса
 *
 * @AmazingClass(true)
 */
class Example
{

    /**
     * Это свойство с особенностью
     *
     * @SpecialFeature
     */
    protected $someProperty;

    /**
     * Это метод
     *
     * @SpecialFeature
     */
    public function someMethod()
    {
        // ...
    }
}
```

В примере выше мы показали аннотации в комментариях, имеющие следующий синтаксис:

`@Annotation-Name[(param1, param2, ...)]`

Аннотации также могут быть помещены в любую часть блока документации:

```
<?php

/**
 * Это свойство с особенностью
 *
 * @SpecialFeature
 *
 * Еще комментарии
 *
 * @AnotherSpecialFeature(true)
 */
```

Парсер является очень гибким инструментом, поэтому следующий блок документации также является правильным:

```
<?php

/**
 * Это свойство с особенностью @SpecialFeature(
someParameter="the value", false
)
Еще комментарии @AnotherSpecialFeature(true) @MoreAnnotations
**/
```

Тем не менее, рекомендуется помещать аннотации в конце блоков документации, чтобы сделать код более понятным и удобным для поддержки:

```
<?php

/**
 * Это свойство с особенностью
 * Еще комментарии
 *
 * @SpecialFeature({someParameter="the value", false})
 * @AnotherSpecialFeature(true)
 */
```

#### 2.44.1 Чтение аннотаций

Для простого получения аннотаций класса с использованием объектно-ориентированного интерфейса, реализован рефлектор:

```
<?php

$reader = new \Phalcon\Annotations\Adapter\Memory();

//Отразить аннотации в классе Example
$reflector = $reader->get('Example');

//Прочесть аннотации в блоке документации класса
$annotations = $reflector->getClassAnnotations();

//Произвести обход всех аннотаций
foreach ($annotations as $annotation) {

    //Вывести название аннотации
    echo $annotation->getName(), PHP_EOL;
```

```
//Вывести количество аргументов
echo $annotation->numberArguments(), PHP_EOL;

//Вывести аргументы
print_r($annotation->getArguments());
}
```

Процесс чтения аннотаций является очень быстрым. Тем не менее, по причинам производительности, мы рекомендуем использовать адаптер для хранения обработанных аннотаций. Адаптеры кэшируют обработанные аннотации, избегая необходимости в их разборе снова и снова.

*Phalcon\Annotations\Adapter\Memory* был использован в примере выше. Этот адаптер кэширует аннотации только в процессе работы, поэтому он более подходит для разработки. Существуют и другие адаптеры, которые можно использовать, когда приложение используется в продакшене.

## 2.44.2 Типы аннотаций

Аннотации могут иметь или не иметь параметров. Параметры могут быть простыми литералами (строкой, числом, булевым типом, null), массивом, хешированным списком или другими аннотациями:

```
<?php

/**
 * Простая аннотация
 *
 * @SomeAnnotation
 */

/**
 * Аннотация с параметрами
 *
 * @SomeAnnotation("hello", "world", 1, 2, 3, false, true)
 */

/**
 * Аннотация с именованными параметрами
 *
 * @SomeAnnotation(first="hello", second="world", third=1)
 * @SomeAnnotation(first: "hello", second: "world", third: 1)
 */

/**
 * Передача массива
 *
 * @SomeAnnotation([1, 2, 3, 4])
 * @SomeAnnotation([1, 2, 3, 4])
 */

/**
 * Передача хеша в качестве параметра
 *
 * @SomeAnnotation({first=1, second=2, third=3})
 * @SomeAnnotation({'first'=1, 'second'=2, 'third'=3})
 * @SomeAnnotation({'first': 1, 'second': 2, 'third': 3})
 * @SomeAnnotation(['first': 1, 'second': 2, 'third': 3])
 */
```

```

/**
 * Вложенные массивы/хеши
 *
 * @SomeAnnotation({"name": "SomeName", "other": [
 *     "foo1": "bar1", "foo2": "bar2", {1, 2, 3},
 * ]})
 */

/**
 * Вложенные аннотации
 *
 * @SomeAnnotation(first=@AnotherAnnotation(1, 2, 3))
 */

```

### 2.44.3 Практическое использование

Далее мы разберем несколько примеров по использованию аннотаций в PHP приложениях:

#### Кэширование с помощью аннотаций

Давайте представим, что у нас есть контроллер и разработчик хочет сделать плагин, который автоматически запускает кэширование если последнее запущенное действие было помечено как имеющее возможность кэширования. Прежде всего, мы зарегистрируем плагин в сервисе Dispatcher, чтобы получать уведомление при выполнении маршрута:

```

<?php

$di['dispatcher'] = function() {

    $eventsManager = new \Phalcon\Events\Manager();

    //Привязать плагин к событию 'dispatch'
    $eventsManager->attach('dispatch', new CacheEnablerPlugin());

    $dispatcher = new \Phalcon\Mvc\Dispatcher();
    $dispatcher->setEventsManager($eventsManager);
    return $dispatcher;
};

```

CacheEnablerPlugin это плагин, который перехватывает каждое запущенное действие в диспетчере, включая кэш если необходимо:

```

<?php

/**
 * Включение кэша для представления, если
 * последнее запущенное действие имело аннотацию @Cache
 */
class CacheEnablerPlugin extends \Phalcon\Mvc\User\Plugin
{

    /**
     * Это событие запускается перед запуском каждого маршрута в диспетчере
     *
     */
    public function beforeExecuteRoute($event, $dispatcher)

```

```
{  
  
    //Разбор аннотаций в текущем запущенном методе  
    $annotations = $this->annotations->getMethod(  
        $dispatcher->getActiveController(),  
        $dispatcher->getActiveMethod()  
    );  
  
    //Проверить, имеет ли метод аннотацию 'Cache'  
    if ($annotations->has('Cache')) {  
  
        //Метод имеет аннотацию 'Cache'  
        $annotation = $annotations->get('Cache');  
  
        //Получить время жизни кэша  
        $lifetime = $annotation->getNamedParameter('lifetime');  
  
        $options = array('lifetime' => $lifetime);  
  
        //Проверить, есть ли определенный пользователем ключ кэша  
        if ($annotation->hasNamedParameter('key')) {  
            $options['key'] = $annotation->getNamedParameter('key');  
        }  
  
        //Включить кэш для текущего метода  
        $this->view->cache($options);  
    }  
  
}  
  
}  
  
}
```

Теперь мы можем использовать аннотации в контроллере:

```
<?php  
  
class NewsController extends \Phalcon\Mvc\Controller  
{  
  
    public function indexAction()  
{  
  
    }  
  
    /**  
     * Это комментарий  
     *  
     * @Cache(lifetime=86400)  
     */  
    public function showAllAction()  
{  
        $this->view->article = Articles::find();  
    }  
  
    /**  
     * Это комментарий  
     *  
     * @Cache(key="my-key", lifetime=86400)  
     */
```

```

public function showAction($slug)
{
    $this->view->article = Articles::findFirstByTitle($slug);
}

}

```

#### Выбор шаблона для отображения

В данном примере мы будем использовать аннотации для того, чтобы сказать объекту класса *Phalcon\ Mvc\ View\ Simple*, что шаблон должен быть отображен, как только закончится выполнение текущего действия.

#### 2.44.4 Адаптеры аннотация

Компонент поддерживает адаптеры с возможностью кэширования проанализированных аннотаций. Это позволяет увеличивать производительность в боевом режиме и моментальное обновление данных при разработке и тестировании.

На- зва- ние	Описание	API
Memory	Аннотации в этом случае хранятся в памяти до завершения запроса. При перезагрузке страницы разбор будет осуществлён заново. Идеально для стадии разработки.	<i>Phalcon\Annotations\Adapter\Memory</i>
Files	Разобранные аннотации хранятся в PHP-файлах, увеличивая производительность без необходимости постоянно анализа.	<i>Phalcon\Annotations\Adapter\Files</i>
APC	Рекомендуется совместное использование с кэшированием байт-кода. Разобранные аннотации хранятся в APC-кэше, самый быстрый адаптер.	<i>Phalcon\Annotations\Adapter\Apc</i>
XCache	Разобранные аннотации хранятся в XCache-кэше. Также является быстрым адаптером.	<i>Phalcon\Annotations\Adapter\Xcache</i>

#### Создание собственных адаптеров

Для создания адаптера необходимо реализовать интерфейс *Phalcon\Annotations\AdapterInterface*

#### 2.44.5 Внешние источники

- Обучение: Creating a custom model's initializer with Annotations

### 2.45 Консольные приложения

CLI приложения выполняются в командной строке. Они часто используются для работы скриптов с долгим временем выполнения, командных утилит и т.п.

## 2.45.1 Структура

Минимальная структура приложений CLI будет выглядеть следующим образом:

- app/config/config.php
- app/tasks/MainTask.php
- app/cli.php <– основной загрузочный файл

## 2.45.2 Создание загрузочного файла

Как и в обычных приложениях MVC, загрузочный файл используется для загрузки приложения. Вместо загрузочного файла index.php, как в веб-приложениях, мы используем cli.php.

Ниже приведен образец загрузочного файлов, который используется для этого примера.

```
<?php

use Phalcon\DI\FactoryDefault\CLI as CliDI,
    Phalcon\CLI\Console as ConsoleApp;

define('VERSION', '1.0.0');

// Используем стандартный для CLI контейнер зависимостей
$di = new CliDI();

// Определяем путь к каталогу приложений
defined('APPLICATION_PATH')
|| define('APPLICATION_PATH', realpath(dirname(__FILE__)));

/**
 * Регистрируем автозагрузчик, и скажем ему, чтобы зарегистрировал каталог задач
 */
$loader = new \Phalcon\Loader();
$loader->registerDirs(
    array(
        APPLICATION_PATH . '/tasks',
    )
);
$loader->register();

// Загружаем файл конфигурации, если он есть
if(is_readable(APPLICATION_PATH . '/config/config.php')) {
    $config = include APPLICATION_PATH . '/config/config.php';
    $di->set('config', $config);
}

// Создаем консольное приложение
$console = new ConsoleApp();
$console->setDI($di);

/**
 * Определяем консольные аргументы
 */
$arguments = array();
$params = array();

foreach($argv as $k => $arg) {
```

```

if($k == 1) {
    $arguments['task'] = $arg;
} elseif($k == 2) {
    $arguments['action'] = $arg;
} elseif($k >= 3) {
    $params[] = $arg;
}
}

if(count($params) > 0) {
    $arguments['params'] = $params;
}

// определяем глобальные константы для текущей задачи и действия
define('CURRENT_TASK', (isset($argv[1]) ? $argv[1] : null));
define('CURRENT_ACTION', (isset($argv[2]) ? $argv[2] : null));

try {
    // обрабатываем входящие аргументы
    $console->handle($arguments);
}
catch (\Phalcon\Exception $e) {
    echo $e->getMessage();
    exit(255);
}

```

Эта часть кода может быть запущена с помощью команды:

```
$ php app/cli.php
```

```
This is the default task and the default action
```

### 2.45.3 Задачи

Принцип работы задач похож на работу контролеров. Любое приложение CLI нуждается, по крайней мере, в MainTask и mainAction, и каждая задача должна иметь mainAction, который будет выполняться, если действие не задано явно.

Ниже приведен пример задачи из файла ‘app/tasks/MainTask.php’:

```
<?php

class mainTask extends \Phalcon\CLI\Task
{

    public function mainAction() {
        echo "\nThis is the default task and the default action \n";
    }

}
```

### 2.45.4 Обработка параметров в Action

Имеется возможность передавать параметры в Action, код для этого уже присутствует в образце загрузочного файла.

Если вы запустите приложение со следующими параметрами и Action:

```
<?php

class mainTask extends \Phalcon\CLI\Task
{

    public function mainAction() {
        echo "\nThis is the default task and the default action \n";
    }

    /**
     * @param array $params
     */
    public function testAction(array $params) {
        echo sprintf('hello %s', $params[0]) . PHP_EOL;
        echo sprintf('best regards, %s', $params[1]) . PHP_EOL;
    }
}

$ php app/cli.php main test world universe

hello world
best regards, universe
```

## 2.45.5 Запуск цепочки команд

Вы также можете запустить цепочку задач, для этого вы должны добавить саму консоль в контейнер зависимостей:

```
<?php

$di->setShared('console', $console);

try {
    // обрабатываем входящие аргументы
    $console->handle($arguments);
}
```

Затем, вы сможете использовать консоль внутри любой задачи. Ниже приведен пример модифицированного MainTask.php:

```
<?php

class MainTask extends \Phalcon\CLI\Task {

    public function mainAction() {
        echo "\nThis is the default task and the default action \n";

        $this->console->handle(array(
            'task' => 'main',
            'action' => 'test'
        ));
    }

    public function testAction() {
        echo '\nI will get printed too!\n';
    }
}
```

```
}
```

Тем не менее, лучшей идеей будет реализовать свой класс, расширяющий PhalconCLITask, и реализовать такую логику там.

## 2.46 Очереди

Выполнение работы, такой как обработка видео, изменения размера изображения или рассылка электронной почты, не подходит для выполнения в Интернете или в реальном времени, потому что это может замедлить время загрузки страниц, влияющее на впечатление пользователей.

Лучшим решением здесь являются фоновые процессы. Веб-приложение должно поставить работу в очередь и ждать, когда она будет выполнена.

Хотя вы можете найти более сложные расширения PHP для реализации очередей в ваших приложениях (такие как RabbitMQ), Phalcon предоставляет клиент для Beanstalk, в котором реализация очередей была вдохновлена Memcache. Этот клиент прост, легок, и полностью специализируется на работе очереди.

### 2.46.1 Добавление заданий в очередь

После подключения к Bens, вы можете добавлять столько заданий сколько необходимо. Разработчик может определить структуру сообщения в соответствии с потребностями приложения:

```
<?php

// Подключение к очереди
$queue = new Phalcon\Queue\Beanstalk(array(
    'host' => '192.168.0.21'
));

// Добавляем задание
$queue->put(array('processVideo' => 4871));
```

Доступные варианты подключения:

Опция	Описание	По умолчанию
host	IP где расположен сервер Beanstalk	127.0.0.1
port	Порт которой слушает Beanstalk	11300

В приведенном выше примере мы сохранили сообщение, которое позволит фоновому заданию обработать видео. Сообщение сохраняется в очереди немедленно и не имеет определенного времени жизни.

Дополнительные опции как времени для запуска, приоритет и задержка может быть передан в качестве второго параметра:

```
<?php

// Добавление задания с опциями в очередь
$queue->put(
    array('processVideo' => 4871),
    array('priority' => 250, 'delay' => 10, 'ttr' => 3600)
);
```

Доступны следующие опции:

Каждое задание, помещаемое в очередь, возвращает “id задачи”, разработчик может использовать для отслеживания статуса задачи:

```
<?php  
  
$jobId = $queue->put(array('processVideo' => 4871));
```

## 2.46.2 Прием сообщений

После того, как задание помещается в очередь, эти сообщения могут быть получены из фонового задания, которое имеет достаточно времени для выполнения задачи:

```
<?php  
  
while (($job = $queue->peekReady()) !== false) {  
  
    $message = $job->getBody();  
  
    var_dump($message);  
  
    $job->delete();  
}
```

Задания должны быть удалены из очереди, чтобы избежать двойной обработки. Если будут реализованы несколько обработчиков задач, то задачи должны быть защищены от возможности повторного запуска другим обработчиком:

```
<?php  
  
while ($queue->peekReady() !== false) {  
  
    $job = $queue->reserve();  
  
    $message = $job->getBody();  
  
    var_dump($message);  
  
    $job->delete();  
}
```

Наш клиент реализует базовый набор функций предоставляемых Beanstalkd, но достаточный, чтобы позволить вам создавать приложения с реализацией очередей.

## 2.47 Уровень абстракции баз данных

*Phalcon\Db* является компонентом, располагающимся под *Phalcon\ Mvc\ Model*, который обеспечивает уровень моделей во фреймворке. Он состоит из независимых абстракций высокого уровня для баз данных, полностью написанных на С.

Этот компонент позволяет производить манипуляции с базой данных на более низком уровне, чем при использовании традиционных моделей.

Данное руководство не претендует на полную документацию доступных методов и аргументов. Пожалуйста, посетите: [API](#) для подробного изучения.

## 2.47.1 Адаптеры базы данных

Данный компонент позволяет использовать адаптеры для инкапсуляции конкретных деталей системы баз данных. Phalcon использует PDO для подключения к базам данных. Поддерживаются следующие СУБД:

Имя	Описание	API
MySQL	Наиболее часто используемая реляционная система управления базами данных (RDBMS), которая работает как сервер, обеспечивающий многопользовательский доступ к некоторому набору баз данных.	<a href="#">Phalcon\Db\Adapter\Pdo\Mysql</a>
PostgreSQL	PostgreSQL является мощной RDBMS с открытым исходным кодом. PostgreSQL – это более чем 15 лет активного развития и проверенная архитектура, чем и завоевала прочную репутацию за надежность, целостность данных и точность.	<a href="#">Phalcon\Db\Adapter\Pdo\Postgresql</a>
SQLite	Библиотека SQLite реализует автономную, бессерверную, не требующую конфигурации и при этом поддерживающую транзакции базу данных на основе языка SQL.	<a href="#">Phalcon\Db\Adapter\Pdo\Sqlite</a>
Oracle	Oracle является объектно-реляционной системой управления базами данных, производится и продается компанией Oracle Corporation.	<a href="#">Phalcon\Db\Adapter\Pdo\Oracle</a>

### Реализации собственных адаптеров

Интерфейс [Phalcon\Db\AdapterInterface](#) должен быть реализован для того, чтобы создать свой собственный адаптер базы данных или расширить существующий.

## 2.47.2 Диалекты баз данных

Приложение инкапсулирует специфические детали каждого компонента баз данных в диалектах. Которые в свою очередь предоставляют адаптером общие функции и генератор SQL.

Имя	Описание	API
MySQL	Специфичные диалекты SQL для MySQL	<a href="#">Phalcon\Db\Dialect\Mysql</a>
PostgreSQL	Специфичные диалекты SQL для PostgreSQL	<a href="#">Phalcon\Db\Dialect\Postgresql</a>
SQLite	Специфичные диалекты SQL для SQLite	<a href="#">Phalcon\Db\Dialect\Sqlite</a>
Oracle	Специфичные диалекты SQL для Oracle	<a href="#">Phalcon\Db\Dialect\Oracle</a>

### Реализации собственных диалектов

Интерфейс [Phalcon\Db\DialetInterface](#) должен быть реализован для того, чтобы создать свой собственный диалект базы данных или расширить существующий.

## 2.47.3 Подключение к базе данных

Чтобы создать подключение необходимо создать экземпляр класса адаптера. Для этого требуется только массив с параметрами соединения. В приведенном примере ниже показано, как создать соединение с обязательными и необязательными параметрами:

```
<?php
// Обязательные
$config = array(
    "host" => "127.0.0.1",
```

```
"username" => "mike",
"password" => "sigma",
"dbname" => "test_db"
);

// Необязательные
$config["persistent"] = false;

// Создаем соединение
$connection = new \Phalcon\Db\Adapter\Pdo\Mysql($config);

<?php

// Обязательные
$config = array(
    "host" => "localhost",
    "username" => "postgres",
    "password" => "secret1",
    "dbname" => "template"
);

// Необязательные
$config["schema"] = "public";

// Создаем соединение
$connection = new \Phalcon\Db\Adapter\Pdo\Postgresql($config);

<?php

// Обязательные
$config = array(
    "dbname" => "/path/to/database.db"
);

// Создаем соединение
$connection = new \Phalcon\Db\Adapter\Pdo\Sqlite($config);

<?php

// Базовая конфигурация
$config = array(
    'username' => 'scott',
    'password' => 'tiger',
    'dbname' => '192.168.10.145/orcl',
);

// Расширенная конфигурация
$config = array(
    'dbname' => '(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1521)))(CONNECT_DATA=(SERV',
    'username' => 'scott',
    'password' => 'tiger',
    'charset' => 'AL32UTF8',
);

// Создаем соединение
$connection = new \Phalcon\Db\Adapter\Pdo\Oracle($config);
```

## 2.47.4 Настройка дополнительных параметров PDO

Вы можете установить опции PDO во время соединения, передавая параметры ‘options’:

```
<?php

// Создаем соединение с настройками PDO
$connection = new \Phalcon\Db\Adapter\Pdo\Mysql(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "sigma",
    "dbname" => "test_db",
    "options" => array(
        PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES \\'UTF8\\\'",
        PDO::ATTR_CASE => PDO::CASE_LOWER
    )
));
```

## 2.47.5 Извлечение строк

*Phalcon\Db* предоставляет несколько методов для получения строк из таблиц. В данном случае требуется использовать синтаксис SQL используемой СУБД:

```
<?php

$sql = "SELECT id, name FROM robots ORDER BY name";

// Отправляем SQL в базу данных
$result = $connection->query($sql);

// Выводим на экран имя робота
while ($robot = $result->fetch()) {
    echo $robot["name"];
}

// Получаем все строки из таблицы в виде массива
$robots = $connection->fetchAll($sql);
foreach ($robots as $robot) {
    echo $robot["name"];
}

// Получаем только первую строку из таблицы
$robot = $connection->fetchOne($sql);
```

По умолчанию эти методы создают массив с ассоциативными и числовыми индексами. Вы можете изменить это поведение с помощью *Phalcon\Db\Result::setFetchMode()*. Этот метод получает константу, которая определяет, какой тип индекса требуется.

Константа	Описание
<i>Phalcon\Db::FETCH_NUM</i>	Возвращает массив с числовыми индексами
<i>Phalcon\Db::FETCH_ASSOC</i>	Возвращает массив с ассоциативными индексами
<i>Phalcon\Db::FETCH_BOTH</i>	Возвращает массив с ассоциативными и числовыми индексами
<i>Phalcon\Db::FETCH_OBJ</i>	Возвращает объект вместо массива

```
<?php
```

```
$sql = "SELECT id, name FROM robots ORDER BY name";
```

```
$result = $connection->query($sql);

$result->setFetchMode(Phalcon\Db::FETCH_NUM);
while ($robot = $result->fetch()) {
    echo $robot[0];
}
```

`Phalcon\Db::query()` возвращает экземпляр класса `Phalcon\Db\Result\Pdo`. Эти объекты инкапсулируют все методы, которые связаны с возвращаемым набором данных, т.е. перебор набора данных, поиск конкретной записи, получение количества строк в наборе данных и т.д.

```
<?php

$sql = "SELECT id, name FROM robots";
$result = $connection->query($sql);

// Перебор набора данных
while ($robot = $result->fetch()) {
    echo $robot["name"];
}

// Получение третьей строки
$result->seek(2);
$robot = $result->fetch();

// Получение количества строк в наборе данных
echo $result->numRows();
```

## 2.47.6 Подготавливаемые запросы

Подготавливаемые запросы также поддерживается в `Phalcon\Db`. Хотя при ее использовании есть минимальное влияние на производительность, рекомендуется использовать эту методику, чтобы исключить возможность SQL инъекций в вашем коде. Поддерживаются как именованные, так и неименованные псевдопеременные. Связывание параметров может просто быть достигнуто следующим образом:

```
<?php

//Подготовленный запрос с неименованными псевдопеременными
$sql     = "SELECT * FROM robots WHERE name = ? ORDER BY name";
$result = $connection->query($sql, array("Wall-E"));

//Подготовленный запрос с именованными псевдопеременными
$sql     = "INSERT INTO `robots`(`name`, year) VALUES (:name, :year)";
$success = $connection->query($sql, array("name" => "Astro Boy", "year" => 1952));
```

## 2.47.7 Вставка/Обновление/Удаление строк

Вставлять, обновлять и удалять строки вы можете с помощью стандартного SQL запроса или использовать методы, предоставляемые классом:

```
<?php

// Вставка с помощью стандартного SQL запроса
$sql     = "INSERT INTO `robots`(`name`, `year`) VALUES ('Astro Boy', 1952)";
$success = $connection->execute($sql);
```

```

// с помощью подготовленного запроса
$sql      = "INSERT INTO `robots`(`name`, `year`) VALUES (?, ?)";
$success = $connection->execute($sql, array('Astroy Boy', 1952));

// Динамическое создание запроса с помощью метода класса
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

// Обновление с помощью стандартного SQL запроса
$sql      = "UPDATE `robots` SET `name` = 'Astro boy' WHERE `id` = 101";
$success = $connection->execute($sql);

// с помощью подготовленного запроса
$sql      = "UPDATE `robots` SET `name` = ? WHERE `id` = ?";
$success = $connection->execute($sql, array('Astroy Boy', 101));

// Динамическое создание запроса с помощью метода класса
$success = $connection->update(
    "robots",
    array("name"),
    array("New Astro Boy"),
    "id = 101"
);

// Удаление с помощью стандартного SQL запроса
$sql      = "DELETE `robots` WHERE `id` = 101";
$success = $connection->execute($sql);

// с помощью подготовленного запроса
$sql      = "DELETE `robots` WHERE `id` = ?";
$success = $connection->execute($sql, array(101));

// Динамическое создание запроса с помощью метода класса
$success = $connection->delete("robots", "id = 101");

```

## 2.47.8 Транзакции и вложенные транзакции

Работа с транзакциями поддерживается с помощью PDO. Манипуляции с данными внутри транзакции часто увеличивает скорость работы базы данных на большинстве систем.

```

<?php

try {

    // Начало новой транзакции
    $connection->begin();

    // Выполнение нескольких команд SQL
    $connection->execute("DELETE `robots` WHERE `id` = 101");
    $connection->execute("DELETE `robots` WHERE `id` = 102");
    $connection->execute("DELETE `robots` WHERE `id` = 103");

    // Фиксируем изменения в транзакции, если все хорошо.

```

```
$connection->commit();  
  
} catch(Exception $e) {  
    // В случае исключения откатываем все изменения  
    $connection->rollback();  
}
```

В дополнение к стандартным транзакциям, Phalcon\Db обеспечивает встроенную поддержку [вложенных транзакций](#) (если база данных поддерживает их). Когда вы вызываете метод `begin()` во второй раз – создается вложенная транзакция:

```
<?php  
  
try {  
  
    // Начало новой транзакции  
    $connection->begin();  
  
    // Выполнение нескольких команд SQL  
    $connection->execute("DELETE 'robots' WHERE 'id' = 101");  
  
    try {  
  
        // Начало вложенной транзакции  
        $connection->begin();  
  
        // Выполнение нескольких команд SQL во вложенной транзакции  
        $connection->execute("DELETE 'robots' WHERE 'id' = 102");  
        $connection->execute("DELETE 'robots' WHERE 'id' = 103");  
  
        // Создаем точку сохранения  
        $connection->commit();  
  
    } catch(Exception $e) {  
        // В случае исключения откатываем все изменения  
        $connection->rollback();  
    }  
  
    // Продолжаем выполнение нескольких команд SQL  
    $connection->execute("DELETE 'robots' WHERE 'id' = 104");  
  
    // Фиксируем изменения в транзакции, если все хорошо.  
    $connection->commit();  
  
} catch(Exception $e) {  
    // В случае исключения откатываем все изменения  
    $connection->rollback();  
}
```

## 2.47.9 События базы данных

*Phalcon\Db* способен передавать *EventsManager*, если оно есть. Некоторые события при возвращении булева значения ‘false’ могут остановить выполняемую операцию. Поддерживаются следующие события:

Название	Когда срабатывает	Может остановить работу?
afterConnect	После успешного подключения к БД	Нет
beforeQuery	Перед отправкой SQL в БД	Да
afterQuery	После отправки запроса в БД	Нет
beforeDisconnect	Перед закрытием временного соединения с БД	Нет
beginTransaction	Перед началом транзакции	Нет
rollbackTransaction	Перед откатом изменений произведенных в транзакции	Нет
commitTransaction	Перед фиксацией транзакции	Нет

Привязать менеджер событий к соединению просто, *Phalcon\Db* будет вызывать событие с именем “db”:

```
<?php
```

```
use Phalcon\Events\Manager as EventsManager,
    \Phalcon\Db\Adapter\Pdo\Mysql as Connection;

$eventsManager = new EventsManager();

// Прослушать все события базы данных
$eventsManager->attach('db', $dbListener);

$connection = new Connection(array(
    "host" => "localhost",
    "username" => "root",
    "password" => "secret",
    "dbname" => "invo"
));

// Назначаем менеджер событий экземпляру адаптера БД
$connection->setEventsManager($eventsManager);
```

Иметь возможность остановить выполнение SQL очень полезно, если вы хотите осуществить последнюю проверку SQL на наличие SQL инъекций:

```
<?php
```

```
$eventsManager->attach('db:beforeQuery', function($event, $connection) {

    // Проверка на наличие вредоносных ключевых слов в SQL
    if (preg_match('/DROP|ALTER/i', $connection->getSQLStatement())) {
        // DROP / ALTER операции не разрешено использовать в приложении,
        // это должно быть SQL инъекция!
        return false;
    }

    //Все хорошо
    return true;
});
```

## 2.47.10 Профилирование запросов SQL

*Phalcon\Db* включает в себя компонент профилирования SQL запросов под названием *Phalcon\Db\Profiler*, который используется для анализа производительности запросов к базе данных для того, чтобы диагностировать проблему с производительностью и обнаружить узкие места.

Профилировать базу данных легко с помощью *Phalcon\Db\Profiler*:

```
<?php

use Phalcon\Events\Manager as EventsManager,
    Phalcon\Db\Profiler as DbProfiler;

$eventsManager = new EventsManager();

$profiler = new DbProfiler();

// Слушаем все события БД
$eventsManager->attach('db', function($event, $connection) use ($profiler) {
    if ($event->getType() == 'beforeQuery') {
        // Запуск профайлера с текущим соединением
        $profiler->startProfile($connection->getSQLStatement());
    }
    if ($event->getType() == 'afterQuery') {
        // Остановка текущего профайлера
        $profiler->stopProfile();
    }
});

// Назначаем менеджер событий соединению
$connection->setEventsManager($eventsManager);

$sql = "SELECT buyer_name, quantity, product_name "
    . "FROM buyers "
    . "LEFT JOIN products ON buyers.pid = products.id";

// Выполняем SQLзапрос
$connection->query($sql);

// Получаем последний профиль в профайльере
$profile = $profiler->getLastProfile();

echo "SQL Statement: ", $profile->getSQLStatement(), "\n";
echo "Start Time: ", $profile->getInitialTime(), "\n";
echo "Final Time: ", $profile->getFinalTime(), "\n";
echo "Total Elapsed Time: ", $profile->getTotalElapsedSeconds(), "\n";
```

Вы также можете создать свой собственный компонент профилирования на основе *Phalcon\Db\Profiler* для записи статистики запросов к БД в режиме реального времени:

```
<?php

use Phalcon\Events\Manager as EventsManager,
    Phalcon\Db\Profiler as Profiler,
    Phalcon\Db\Profiler\Item as Item;

class DbProfiler extends Profiler
{

    /**
     * Выполняется перед отправкой SQL запроса на сервер БД
     */
    public function beforeStartProfile(Item $profile)
    {
        echo $profile->getSQLStatement();
    }
}
```

```

/**
 * Выполняется после отправки SQL запроса на сервер БД
 */
public function afterEndProfile(Item $profile)
{
    echo $profile->getTotalElapsedSeconds();
}

// Создание менеджера событий
$eventsManager = new EventsManager();

// Создание слушателя
$dbProfiler = new DbProfiler();

// Прикрепляем слушателя ко всем событиям базы данных
$eventsManager->attach('db', $dbProfiler);

```

### 2.47.11 Логирование SQL запросов

Использование компонентов высокого уровня абстракции для доступа к базам данных, таких как *Phalcon\|Db*, усложняет понимание того, какие запросы отправляются в базу данных. *Phalcon\|Logger* взаимодействует с *Phalcon\|Db*, обеспечивая возможность логирования в слое абстракции базы данных.

```

<?php

use Phalcon\Logger,
    Phalcon\Events\Manager as EventsManager,
    Phalcon\Logger\Adapter\File as FileLogger;

$eventsManager = new EventsManager();

$logger = new FileLogger("app/logs/db.log");

// Слушаем все события БД
$eventsManager->attach('db', function($event, $connection) use ($logger) {
    if ($event->getType() == 'beforeQuery') {
        $logger->log($connection->getSQLStatement(), Logger::INFO);
    }
});

// Назначаем менеджер событий соединению
$connection->setEventsManager($eventsManager);

// Выполняем несколько SQL запросов
$connection->insert(
    "products",
    array("Hot pepper", 3.50),
    array("name", "price")
);

```

Упомянутый выше файл *app/logs/db.log* будет содержать что-то похожее на это:

```
[Sun, 29 Apr 12 22:35:26 -0500] [DEBUG] [Resource Id #77] INSERT INTO products
(name, price) VALUES ('Hot pepper', 3.50)
```

## Реализация собственного логера

Вы можете реализовать свой собственный класс логера запросов к базе данных, путем создания класса, который реализует единственный метод, именуемый «log». Метод должен принимать строку в качестве первого аргумента. Затем Вы можете передать ваш объект логера в метод Phalcon\Db::setLogger(), после чего любые выполняемые запросы SQL будут вызывать этот метод для логирования результата запроса.

### 2.47.12 Описание Таблиц / Представлений

*Phalcon\Db* также предоставляет методы для получения подробной информации о таблицах и представлениях:

```
<?php

// Получаем таблицы из базы данных test_db
$tables = $connection->listTables("test_db");

// Есть ли таблица 'robots' в базе данных?
$exists = $connection->tableExists("robots");

// Получаем имена, типы данных и свойства полей таблицы 'robots'
$fields = $connection->describeColumns("robots");
foreach ($fields as $field) {
    echo "Column Type: ", $field["Type"];
}

// Получаем индексы таблицы 'robots'
$indexes = $connection->describeIndexes("robots");
foreach ($indexes as $index) {
    print_r($index->getColumns());
}

// Получаем внешние ключи на таблицу 'robots'
$references = $connection->describeReferences("robots");
foreach ($references as $reference) {
    // Выводим на экран ссылаемые столбцы
    print_r($reference->getReferencedColumns());
}
```

Таблица описания очень похожа на результат команды “DESCRIBE” в MySQL, она содержит следующую информацию:

Индексы	Описание
Field	Имя поля
Type	Тип столбца
Key	Является ли столбец частью первичного ключа или индексом?
Null	Допускается ли значение NULL.

Методы для получения сведений о представлениях также реализованы для всех поддерживаемых баз данных:

```
<?php

// Получить все представления из базы данных 'test_db'
$tables = $connection->listViews("test_db");
```

```
// Есть ли представление 'robots' в базе данных?
$exists = $connection->viewExists("robots");
```

### 2.47.13 Создание / Изменение / удаление таблиц

Различные системы баз данных (MySQL, PostgreSQL и др.) предоставляют возможность создавать, изменять или удалять таблицы с использованием таких команд как CREATE, ALTER или DROP. Синтаксис SQL отличается в зависимости от того, какая база данных используется системой. *Phalcon\|Db* предлагает единый интерфейс для изменения таблиц, без необходимости дифференцировать синтаксис SQL на основании системы хранения данных.

#### Создание таблиц

В следующем примере показано, как создать таблицу:

```
<?php

use \Phalcon\Db\Column as Column;

$connection->createTable(
    "robots",
    null,
    array(
        "columns" => array(
            new Column("id",
                array(
                    "type"      => Column::TYPE_INTEGER,
                    "size"      => 10,
                    "notNull"   => true,
                    "autoIncrement" => true,
                )
            ),
            new Column("name",
                array(
                    "type"      => Column::TYPE_VARCHAR,
                    "size"      => 70,
                    "notNull"   => true,
                )
            ),
            new Column("year",
                array(
                    "type"      => Column::TYPE_INTEGER,
                    "size"      => 11,
                    "notNull"   => true,
                )
            )
        )
    );
);
```

*Phalcon\|Db::createTable()* принимает ассоциативный массив описывающий таблицу. Столбцы определяются классом *Phalcon\|Db\|Column*. В таблице ниже показаны варианты, доступные для определения столбца:

Опция	Описани	Необязательный?
“type”	Тип столбца. Должен быть константой быть Phalcon\Db\Column constant (см. ниже список)	Нет
“primary”	True, если столбец является частью первичного ключа таблицы	Yes
“size”	Некоторые типы столбцов, такие как VARCHAR или INTEGER, могут иметь определенный размер	Yes
“scale”	DECIMAL или NUMBER столбцы могут указывать точность (до какого десятичного знака требуется хранить числа).	Yes
“unsigned”	INTEGER столбцы могут быть знаковыми или беззнаковыми. Эта опция не распространяется на другие типы столбцов.	Yes
“notNull”	Может ли столбец хранить нулевые значения?	Yes
“autoIncrement”	С помощью этой опции, столбец заполняется автоматически с автоинкрементным целым. Только один столбец в таблице может иметь этот атрибут.	Yes
“bind”	Одна из BIND_TYPE_* констант говорящая, как колонки должны быть привязаны перед сохранением.	Yes
“first”	Колонки должны быть расположены на первой позиции в порядке столбцов	Yes
“after”	Колонка должна быть расположена после указанного столбца	Yes

Phalcon\Db поддерживает следующие типы столбцов базы данных:

- Phalcon\Db\Column::TYPE\_INTEGER
- Phalcon\Db\Column::TYPE\_DATE
- Phalcon\Db\Column::TYPE\_VARCHAR
- Phalcon\Db\Column::TYPE\_DECIMAL
- Phalcon\Db\Column::TYPE\_DATETIME
- Phalcon\Db\Column::TYPE\_CHAR
- Phalcon\Db\Column::TYPE\_TEXT

Ассоциативный массив, переданный в Phalcon\Db::createTable() может иметь возможных ключей:

Индекс	Описание	Необязательный?
“columns”	Массив с набором столбцов таблицы определен с <a href="#">Phalcon\Db\Column</a>	Нет
“indexes”	Массив с набором индексов таблицы, определенные с <a href="#">Phalcon\Db\Index</a>	Да
“reference”	Массив с набором ссылок на таблицы (внешние ключи), определенный с <a href="#">Phalcon\Db\Reference</a>	Да
“options”	Массив с набором опций для создания таблицы. Эти опции часто связаны с системой базы данных, в которых миграции был сгенерирован.	Да

## Изменение таблиц

Если ваше приложение растет, вам, возможно, потребуется вносить изменения в базу данных, как часть рефакторинга или добавление нового функционала. Не все базы данных позволяют изменять существующие столбцы или добавить столбцы между двумя существующими. [Phalcon\Db](#) ограничено этими особенностями реализаций.

<?php

```
use Phalcon\Db\Column as Column;
```

```
// Добавляем новый столбец
$connection->addColumn("robots", null,
    new Column("robot_type", array(
        "type"      => Column::TYPE_VARCHAR,
        "size"      => 32,
        "notNull"   => true,
        "after"     => "name"
    ))
);

// Изменение существующего столбца
$connection->modifyColumn("robots", null, new Column("name", array(
    "type"      => Column::TYPE_VARCHAR,
    "size"      => 40,
    "notNull"   => true,
)));
;

// Удаление столбца "name"
$connection->deleteColumn("robots", null, "name");
```

## Удаление таблицы

Пример удаления таблицы:

```
<?php

// Удаление таблицы "robots" из активной базы данных
$connection->dropTable("robots");

//Удаление таблицы "robots" из базы данных "machines"
$connection->dropTable("robots", "machines");
```

## 2.48 Интернационализация

Phalcon является расширением для PHP, написанным на языке С. Существует PECL-расширение `intl`, которое предоставляет функции, обеспечивающие поддержку интернационализации в PHP. Начиная с PHP 5.4/5.5, это расширение поставляется вместе с PHP. Документация по нему может быть найдена на страницах официального Руководства по PHP.

Phalcon не предоставляет функциональности этого расширения, так как создание такого компонента являлось бы повторением существующего кода.

В примере ниже мы покажем вам как воспользоваться функциональностью расширения `intl` в приложениях, написанных с использованием Phalcon.

Данное руководство не ставит перед собой цель подробно документировать расширение `intl`. Для справок, пожалуйста, обратитесь к [документации](#) этого расширения.

### 2.48.1 Выяснение наиболее подходящей локали

Существует несколько способов для выяснения наиболее подходящей локали с помощью `intl`. Одним из них является проверка HTTP-заголовка “Accept-Language”:

```
<?php

$locale = Locale::acceptFromHttp($_SERVER["HTTP_ACCEPT_LANGUAGE"]);

// Локалью может быть что-то вроде "ru_RU" или "ru"
echo $locale;
```

Метод выше возвращает идентифицированную локаль. Она используется для получения языковых, культурных или региональных особенностей поведения с использованием API класса Locale. Примерами идентификаторов локали являются:

- en-US (Английский, США)
- ru-RU (Русский, Россия)
- zh-Hant-TW (Китай, Традиционный китайский, Тайвань)
- fr-CA, fr-FR (Французский для Канады и Франции соответственно)

## 2.48.2 Форматирование сообщений на основании локали

Неотъемлемой частью разработки локализованного приложения является создание объединенных, независимых от языка сообщений. `MessageFormatter` позволяет создавать такие сообщения.

Печать форматированных чисел на основе разных локалей:

```
<?php

// Выведем € 4 560
formatter = new MessageFormatter("fr_FR", "€ {0, number, integer}");
echo $formatter->format(array(4560));

// Выведем USD$ 4,560.5
formatter = new MessageFormatter("en_US", "USD$ {0, number}");
echo $formatter->format(array(4560.50));

// Выведем ARS$ 1.250,25
formatter = new MessageFormatter("es_AR", "ARS$ {0, number}");
echo $formatter->format(array(1250.25));
```

Форматирование сообщений, используя шаблоны времени и даты:

```
<?php

//Устанавливаем параметры
$time = time();
$values = array(7, $time, $time);

// Выведем "At 3:50:31 PM on Apr 19, 2012, there was a disturbance on planet 7."
$pattern = "At {1, time} on {1, date}, there was a disturbance on planet {0, number}.";
formatter = new MessageFormatter("en_US", $pattern);
echo $formatter->format($values);

// Выведем "À 15:53:01 le 19 avr. 2012, il y avait une perturbation sur la planète 7."
$pattern = "À {1, time} le {1, date}, il y avait une perturbation sur la planète {0, number}.";
formatter = new MessageFormatter("fr_FR", $pattern);
echo $formatter->format($values);
```

### 2.48.3 Сравнение строк с учетом локали

Класс `Collator` предоставляет возможности по сравнению строк, чувствительных к локали, с поддержкой соответствующих правил сравнений. Ниже приведены примеры, демонстрирующие использование этого класса:

```
<?php

// Создаем коллатор, использующий испанскую локаль
$collator = new Collator("es");

// Результат сравнения будет положительный, несмотря на ударение над "о"
$collator->setStrength(Collator::PRIMARY);
var_dump($collator->compare("una canción", "una cancion"));

// Результат сравнения будет отрицательный
$collator->setStrength(Collator::DEFAULT_VALUE);
var_dump($collator->compare("una canción", "una cancion"));
```

### 2.48.4 Транслитерация

Компонент `Transliterator` добавляет возможность транслитерации строк:

```
<?php

$id = "Any-Latin; NFD; [:Nonspacing Mark:] Remove; NFC; [:Punctuation:] Remove; Lower();";
$transliterator = Transliterator::create($id);

$string = "garçon-étudiant-où-L'école";
echo $transliterator->transliterate($string); // garconetudiantoulecole
```

## 2.49 Миграции базы данных

Миграции это удобный для вас способ, структурировано и организовано изменять ваши базы данных.

**Важно:** Миграции доступны через *Phalcon Developer Tools*. Вам потребуется Phalcon версии не ниже 0.5.0 для использования инструментов разработчика. Также рекомендуется использовать PHP версии 5.3.11 или более поздней версии.

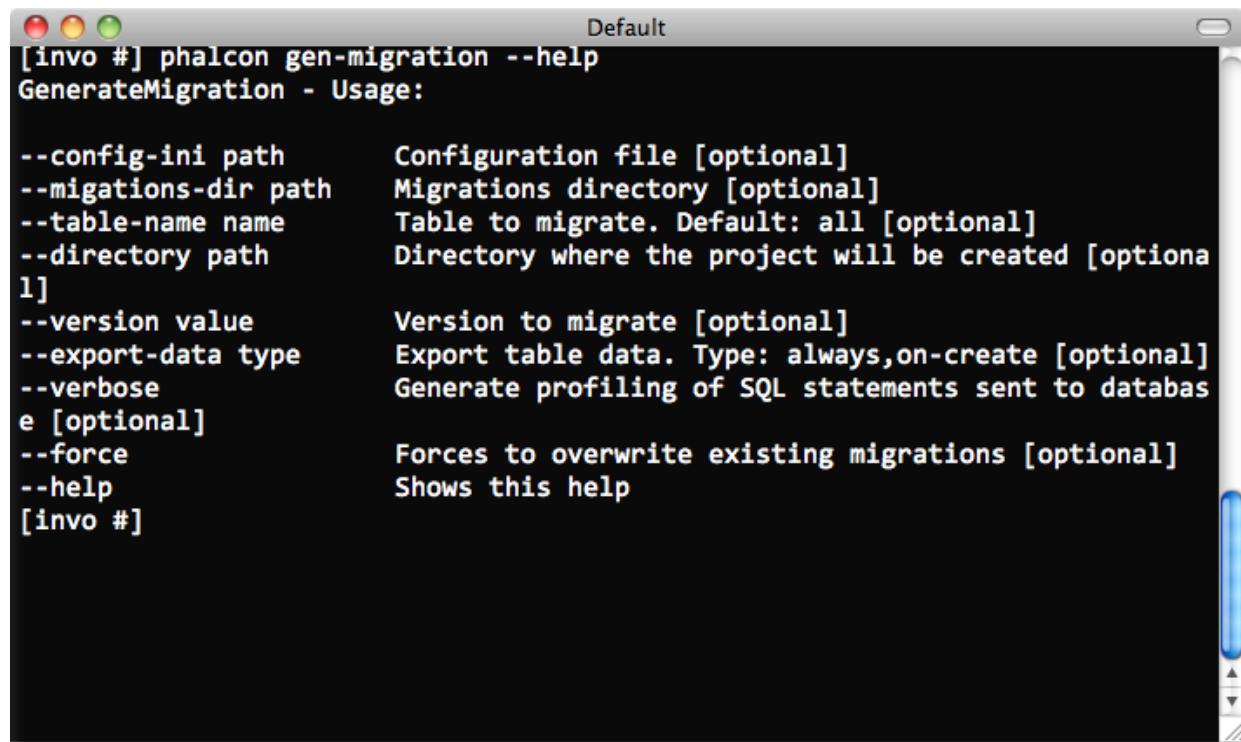
Часто при разработке необходимо вносить изменения на production окружении. Некоторые из этих изменений могут касаться изменений в базе данных: новые столбцы, новые таблицы, удаление индексов и т.д.

При миграции создается набор классов, чтобы описать, как ваша база данных структурирована в данный момент. Эти классы могут использоваться для синхронизации структуры схемы в удаленных базах данных и подготовки вашей базы данных к работе с новыми изменениями, которые реализует ваше приложение. Миграции описывают эти изменения с использованием простого PHP.

### 2.49.1 Дампинг схемы

*Phalcon Developer Tools* предоставляет скрипт для управления миграциями (генерация, запуск и откат).

Доступные опции для генерации миграций:



```

Default
[invo #] phalcon gen-migration --help
GenerateMigration - Usage:

--config-ini path      Configuration file [optional]
--migrations-dir path  Migrations directory [optional]
--table-name name      Table to migrate. Default: all [optional]
--directory path        Directory where the project will be created [optional]
[1]
--version value         Version to migrate [optional]
--export-data type      Export table data. Type: always,on-create [optional]
--verbose               Generate profiling of SQL statements sent to database
e [optional]
--force                 Forces to overwrite existing migrations [optional]
--help                  Shows this help
[invo #]

```

Запуск скрипта без параметров делает простой дамп каждого объекта (таблиц и представлений) из базы данных в классы миграции.

Каждая миграция имеет идентификатор версии связанный с ним. Номер версии позволяет нам определить, является ли миграция старше или новее текущей версии нашей базы данных. Версии также сообщают Phalcon о рабочем состоянии при выполнении миграции.

При генерации миграций, инструкции отображаются в консоли для описания различных этапов миграции и времени выполнения этих операторов. В конце концов, версия миграция будет создана.

По умолчанию *Phalcon Developer Tools* использует директорию *app/migrations* для дампа файлов миграции. Вы можете изменить расположение, установив один из параметров по генерации скрипта. Каждая таблица в базе данных имеет свой соответствующий класс, созданный в отдельном файле директории, ссылающейся на ее версию.

## 2.49.2 Структура класса Migration

Каждый файл содержит уникальный класс, который расширяет *PhalconMvcModelMigration*. Эти классы обычно имеют два метода: *up()* и *down()*. *Up()* выполняет миграцию, а *down()* откатывает ее.

Метод *Up()* также содержит магический метод *morphTable()*. Магия начинается тогда, когда он распознает изменения, требующие синхронизации фактической таблицы в базе данных, приведенные выше.

```
<?php
```

```

use Phalcon\Db\Column as Column;
use Phalcon\Db\Index as Index;
use Phalcon\Db\Reference as Reference;

class ProductsMigration_100 extends \Phalcon\Mvc\Model\Migration
{

```

```
Default
T, TABLES.ENGINE, TABLES.TABLE_COLLATION FROM INFORMATION_SCHEMA
.TABLES WHERE TABLES.TABLE_SCHEMA = "invo" AND TABLES.TABLE_NA
ME = "test" => 1335920161.9368 (0.00032401084899902)
1335920161.9369: DESCRIBE `invo`.`users` => 1335920161.938 (0
.0010280609130859)
1335920161.9383: SHOW INDEXES FROM `invo`.`users` => 13359201
61.9389 (0.00062680244445801)
1335920161.939: SELECT TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME,
REFERENCED_TABLE_SCHEMA, REFERENCED_TABLE_NAME, REFERENCED_COLU
N_NAME FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE WHERE REFER
ENCED_TABLE_NAME IS NOT NULL AND CONSTRAINT_SCHEMA = "invo" AN
D TABLE_NAME = "users" => 1335920161.9403 (0.0012459754943848
)
1335920161.9404: SELECT TABLES.TABLE_TYPE, TABLES.AUTO_INCREMENT
T, TABLES.ENGINE, TABLES.TABLE_COLLATION FROM INFORMATION_SCHEMA
.TABLES WHERE TABLES.TABLE_SCHEMA = "invo" AND TABLES.TABLE_NA
ME = "users" => 1335920161.9407 (0.00038504600524902)
Version 1.0.0 was successfully generated
[invo #]
```

```
Default
[invo #] cd app/migrations/1.0.0/
[1.0.0 #] ls
companies.php           products.php
contact.php              test.php
product_types.php        users.php
[1.0.0 #] ■
```

```
public function up()
{
    $this->morphTable(
        "products",
        array(
            "columns" => array(
                new Column(
                    "id",
                    array(
                        "type"      => Column::TYPE_INTEGER,
                        "size"      => 10,
                        "unsigned"  => true,
                        "notNull"   => true,
                        "autoIncrement" => true,
                        "first"     => true,
                    )
                ),
                new Column(
                    "product_types_id",
                    array(
                        "type"      => Column::TYPE_INTEGER,
                        "size"      => 10,
                        "unsigned"  => true,
                        "notNull"   => true,
                        "after"     => "id",
                    )
                ),
                new Column(
                    "name",
                    array(
                        "type"      => Column::TYPE_VARCHAR,
                        "size"      => 70,
                        "notNull"   => true,
                        "after"     => "product_types_id",
                    )
                ),
                new Column(
                    "price",
                    array(
                        "type"      => Column::TYPE_DECIMAL,
                        "size"      => 16,
                        "scale"     => 2,
                        "notNull"   => true,
                        "after"     => "name",
                    )
                ),
            ),
            "indexes" => array(
                new Index(
                    "PRIMARY",
                    array("id")
                ),
                new Index(
                    "product_types_id",
                    array("product_types_id")
                )
            ),
            "references" => array(
```

```

        new Reference(
            "products_ibfk_1",
            array(
                "referencedSchema" => "invo",
                "referencedTable"   => "product_types",
                "columns"          => array("product_types_id"),
                "referencedColumns" => array("id"),
            )
        )
    ),
    "options" => array(
        "TABLE_TYPE"      => "BASE TABLE",
        "ENGINE"          => "InnoDB",
        "TABLE_COLLATION" => "utf8_general_ci",
    )
)
);
}
}

```

Класс называется “ProductsMigration\_100”. Suffix 100 указывает на версию 1.0.0. morphTable() принимает ассоциативный массив с 4 возможными типами данных:

## Определение столбцов

*Phalcon\Db\Column* используется для определения столбцов таблицы. It encapsulates column related features. Его конструктор принимает в качестве первого параметра имя столбца и массив, описывающий колонки. Доступны следующие опции при описании столбцов:

| Опция           | Описание  | Optional |
|-----------------|---|----------|
| “type”          | Тип столбца. <i>Phalcon_Db_Column</i> должен быть константой (смотрите ниже)  | No       |
| “size”          | Некоторые типы столбцов, как VARCHAR или INTEGER могут иметь определенный размер  | Yes      |
| “scale”         | Столбцы DECIMAL или NUMBER могут иметь разрешение точности, чтобы указать до какого десятичного знака необходимо хранить значение.          | Yes      |
| “unsigned”      | INTEGER столбцы могут быть знаковыми или беззнаковыми. Эта опция не распространяется на другие типы столбцов                                | Yes      |
| “notNull”       | Столбец может хранить нулевые значения?   | Yes      |
| “autoIncrement” | С помощью этого атрибута столбец заполняется автоматически с автоинкрементным целым. Только один столбец в таблице может иметь этот атрибут | Yes      |
| “first”         | Столбец должны быть расположены на первые позиции в порядке столбцов  | Yes      |
| “after”         | Колонка должна быть помещена после указанного столбца   | Yes      |

Миграции базы данных поддерживают следующие типы столбцов базы данных:

- Phalcon\Db\Column::TYPE\_INTEGER
- Phalcon\Db\Column::TYPE\_DATE
- Phalcon\Db\Column::TYPE\_VARCHAR
- Phalcon\Db\Column::TYPE\_DECIMAL
- Phalcon\Db\Column::TYPE\_DATETIME
- Phalcon\Db\Column::TYPE\_CHAR

- Phalcon\Db\Column::TYPE\_TEXT

### Определение индексов

*Phalcon\Db\Index* определяет индексы таблицы. Для создания индекса требуется определить его имя и список столбцов. Заметим, что если любой индекс имеет название PRIMARY, то Phalcon создаст индекс первичного ключа в этой таблице.

### Определение внешних ключей

*Phalcon\Db\Reference* определяет ссылки на таблицы (также называемые внешними ключами). Следующие опции могут быть использованы для определения внешних ключей:

| Индекс             | Описание   | Optional |
|--------------------|--|----------|
| “referencedTable”  | It's auto-descriptive. Содержит имя ссылочной таблицы.   | No       |
| “columns”          | Массив с названием столбцов в таблице, которые имеют ссылки.   | No       |
| “referencedColumn” | Массив с именем столбцов в указанной таблице.  | No       |
| “referencedTable”  | Ссылочной таблицы, может быть, находится в другой схеме или базе данных. Эта опция позволяет вам определить это. | Yes      |

### 2.49.3 Запись миграций

Миграции не только предназначены для “морфинга” таблицы. Миграция является обычным классом PHP, так что вы не ограничены этими функциями. Например, после добавления столбца можно написать код для установки значений этого столбца для существующих записей. Для более подробной информации и примеров отдельных методов, проверьте *database component*.

```
<?php

class ProductsMigration_100 extends \Phalcon\Mvc\Model\Migration
{

    public function up()
    {
        //...
        self::$_connection->insert(
            "products",
            array("Malabar spinach", 14.50),
            array("name", "price")
        );
    }

}
```

### 2.49.4 Запуск миграций

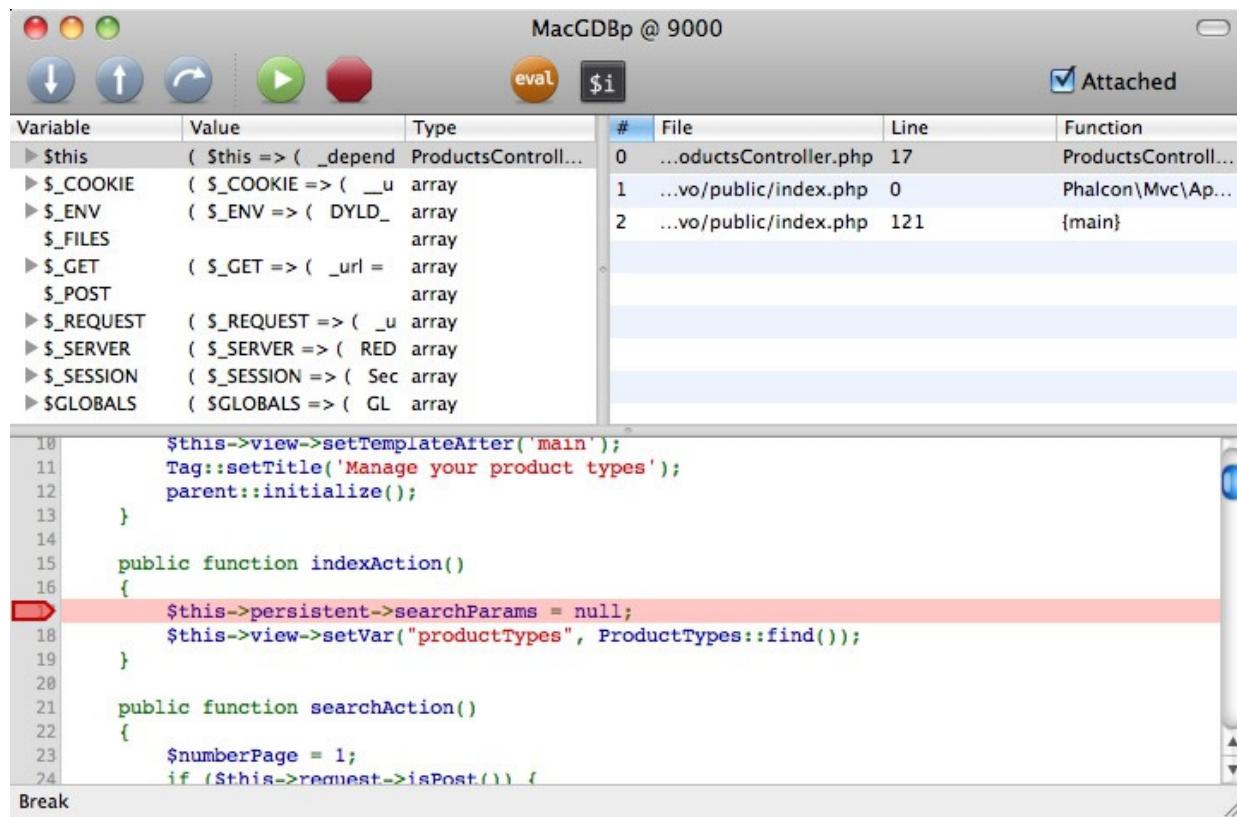
Как только сгенерированные миграции, загруженных на целевой сервер, вы можете легко запустить их, как показано в следующем примере:

В зависимости от того, насколько сильно устаревшей является база данных по миграции, Phalcon может запускать несколько версий миграции в одном процессе миграции. Если указать целевую версию, Phalcon будет запускать нужные миграции пока не достигнет указанной версии.

```
[invo #] phalcon run-migration
```

```
657)
1335932029.8233: SHOW INDEXES FROM `invo`.`products` => 13359
32029.8237 (0.00034308433532715)
1335932029.8246: SELECT COUNT(*) FROM `INFORMATION_SCHEMA`.`TA
BLES` WHERE `TABLE_NAME` = 'test' AND `TABLE_SCHEMA`='invo' =>
1335932029.8249 (0.00030779838562012)
1335932029.825: DESCRIBE `invo`.`test` => 1335932029.8259 (0.
00094413757324219)
1335932029.826: SHOW INDEXES FROM `invo`.`test` => 1335932029
.8269 (0.00085902214050293)
1335932029.8275: SELECT COUNT(*) FROM `INFORMATION_SCHEMA`.`TA
BLES` WHERE `TABLE_NAME` = 'users' AND `TABLE_SCHEMA`='invo' =
> 1335932029.8278 (0.00026392936706543)
1335932029.8279: DESCRIBE `invo`.`users` => 1335932029.8287 (
0.00088977813720703)
1335932029.8289: SHOW INDEXES FROM `invo`.`users` => 13359320
29.8299 (0.00097990036010742)
Version 1.0.0 was successfully migrated
[invo #]
```

## 2.50 Отладка приложений



PHP предлагает набор инструментов для отладки приложений с выводом уведомлений, предупреждений, ошибок и исключений. Класс `Exception class` передает различную информацию о том, где было сгенерировано исключение: файл, линию, сообщение, код ошибки, список вызовов и т.п. ООП системы, такие как Phalcon, в основном используют этот класс в качестве родительского для добавления различного функционала и предоставления информации разработчику или пользователю.

Несмотря на то, что Phalcon написан на языке C, он вызывает методы из пользовательского уровня PHP, обеспечивая возможность отладки и совместимость с другими приложениями.

### 2.50.1 Перехват исключений

Существует основной способ перехвата исключений, через конструкцию `try/catch`:

```

<?php
try {
    //... какой-то код phalcon/php
} catch(\Exception $e) {
}

```

Исключение, перехваченное в этом блоке, попадает в переменную `$e`. А `Phalcon\Exception` унаследован от PHP класса `Exception class` и используется, чтобы понять, является ли исключение из Phalcon или

из PHP.

Все исключение сгенерированные в PHP базируются на классе `Exception class`, и имеют следующий набор элементов:

```
<?php

class Exception
{

    /* Свойства */
    protected string $message;
    protected int $code;
    protected string $file;
    protected int $line;

    /* Методы */
    public __construct ([ string $message = "" [, int $code = 0 [, Exception $previous = NULL ]]])
    final public string getMessage ( void )
    final public Exception getPrevious ( void )
    final public mixed getCode ( void )
    final public string getFile ( void )
    final public int getLine ( void )
    final public array getTrace ( void )
    final public string getTraceAsString ( void )
    public string __toString ( void )
    final private void __clone ( void )

}
```

Получить информацию из `Phalcon\Exception` можно таким же способом, как из `Exception class`:

```
<?php

try {

    //... код приложения ...

} catch(\Exception $e) {
    echo get_class($e), ":", $e->getMessage(), "\n";
    echo " File=", $e->getFile(), "\n";
    echo " Line=", $e->getLine(), "\n";
    echo $e->getTraceAsString();
}
```

Таким образом, можно легко узнать, где было сгенерировано исключение (файл, строка) и какие компоненты участвовали в генерации:

```
PDOException: SQLSTATE[28000] [1045] Access denied for user 'root'@'localhost'
(using password: NO)
File=/Applications/MAMP/htdocs/invo/public/index.php
Line=74
#0 [internal function]: PDO->__construct('mysql:host=loca...', 'root', '', Array)
#1 [internal function]: Phalcon\Db\Adapter\Pdo->connect(Array)
#2 /Applications/MAMP/htdocs/invo/public/index.php(74):
    Phalcon\Db\Adapter\Pdo->__construct(Array)
#3 [internal function]: {closure}()
#4 [internal function]: call_user_func_array(Object(Closure), Array)
#5 [internal function]: Phalcon\DI->factory(Object(Closure), Array)
#6 [internal function]: Phalcon\DI->get('db', Array)
```

```
#7 [internal function]: Phalcon\DI->getShared('db')
#8 [internal function]: Phalcon\Mvc\Model->getConnection()
#9 [internal function]: Phalcon\Mvc\Model::__getOrCreateResultset('Users', Array, true)
#10 /Applications/MAMP/htdocs/invo/app/controllers/SessionController.php(83):
    Phalcon\Mvc\Model::findFirst('email='demo@pha...')
#11 [internal function]: SessionController->startAction()
#12 [internal function]: call_user_func_array(Array, Array)
#13 [internal function]: Phalcon\Mvc\Dispatcher->dispatch()
#14 /Applications/MAMP/htdocs/invo/public/index.php(114): Phalcon\Mvc\Application->handle()
#15 {main}
```

Как вы можете увидеть из вывода исключений, все методы прозрачны, и можно полностью отследить работу приложения, а так же параметры, которые передавались в методы. Метод `Exception::getTrace` предоставляет дополнительную информацию, если необходимо.

## 2.50.2 Компонент отладки

Phalcon предоставляет компонент отладки, который позволяет разработчикам легко находить ошибки, возникающие в приложении, созданным с помощью фреймворка.

Следующий ролик объясняет, как это работает:

Чтобы включить его, вставьте следующие строки в файл загрузки приложения:

```
<?php
```

```
$debug = new \Phalcon\Debug();
$debug->listen();
```

Любой обработчик исключений (`try/catch`) должен быть удален или заблокирован, чтобы позволить этому компоненту самому перехватывать всплывающие исключения.

## 2.50.3 Рефлексия (Reflection)

Любой экземпляр класса в Phalcon предоставляет тоже поведение, что и во всех экземплярах PHP классов. Можно использовать [Reflection API](#) или просто вывести любой объект, чтобы увидеть его состояние:

```
<?php
```

```
$router = new Phalcon\Mvc\Router();
print_r($router);
```

Таким образом, можно узнать всю информацию о любом объекте. Этот пример выводит такую информацию:

```
Phalcon\Mvc\Router Object
(
    [_dependencyInjector:protected] =>
    [_module:protected] =>
    [_controller:protected] =>
    [_action:protected] =>
    [_params:protected] => Array
        (
        )
    [_routes:protected] => Array
```

```
[0] => Phalcon\Mvc\Router\Route Object
(
    [_pattern:protected] => #^/([a-zA-Z0-9\_]+)[/]{}{0,1}#$#
    [_compiledPattern:protected] => #^/([a-zA-Z0-9\_]+)[/]{}{0,1}#$#
    [_paths:protected] => Array
    (
        [controller] => 1
    )

    [_methods:protected] =>
    [_id:protected] => 0
    [_name:protected] =>
)
)

[1] => Phalcon\Mvc\Router\Route Object
(
    [_pattern:protected] => #^/([a-zA-Z0-9\_]+)/([a-zA-Z0-9\_]+)(\.*)*$#
    [_compiledPattern:protected] => #^/([a-zA-Z0-9\_]+)/([a-zA-Z0-9\_]+)(\.*)*$#
    [_paths:protected] => Array
    (
        [controller] => 1
        [action] => 2
        [params] => 3
    )
    [_methods:protected] =>
    [_id:protected] => 1
    [_name:protected] =>
)
)

[_matchedRoute:protected] =>
[_matches:protected] =>
[_wasMatched:protected] =>
[_defaultModule:protected] =>
[_defaultController:protected] =>
[_defaultAction:protected] =>
[_defaultParams:protected] => Array
(
)
```

## 2.50.4 Использование XDebug

**XDebug** великолепный инструмент для отладки PHP приложений. Он так же является дополнением, написанным на языке C, и вы можете использовать его вместе с Phalcon без дополнительной конфигурации или побочных эффектов.

Следующий ролик показывает работу Xdebug с Phalcon:

После того, как вы установите Xdebug, вы сможете использовать свой API, чтобы получить более подробные сведения об исключениях и сообщениях. .. highlights:

Мы настоятельно рекомендуем использовать по крайней мере XDebug версии 2.2.3 для лучшей совместимости с Phalcon

Следующий пример использует `xdebug_print_function_stack` для остановки выполнения программы и вывода стека вызовов:

```
<?php

class SignupController extends \Phalcon\Mvc\Controller
{

    public function indexAction()
    {

    }

    public function registerAction()
    {

        // Запрос переменных из html формы
        $name = $this->request->getPost("name", "string");
        $email = $this->request->getPost("email", "email");

        // Останавливаем выполнение и выводим стек вызовов
        return xdebug_print_function_stack("stop here!");

        $user      = new Users();
        $user->name = $name;
        $user->email = $email;

        // Сохраняем и проверяем на ошибки
        $user->save();
    }

}
```

Xdebug так же покажет локальные переменные в этом экземпляре:

```
Xdebug: stop here! in /Applications/MAMP/htdocs/tutorial/app/controllers/SignupController.php
on line 19
```

Call Stack:

```
0.0383    654600  1. {main}() /Applications/MAMP/htdocs/tutorial/public/index.php:0
0.0392    663864  2. Phalcon\Mvc\Application->handle()
    /Applications/MAMP/htdocs/tutorial/public/index.php:37
0.0418    738848  3. SignupController->registerAction()
    /Applications/MAMP/htdocs/tutorial/public/index.php:0
0.0419    740144  4. xdebug_print_function_stack()
    /Applications/MAMP/htdocs/tutorial/app/controllers/SignupController.php:19
```

Xdebug предоставляет несколько путей для отладки ваших приложений и получения отладочной информации. Вы можете ознакомиться с [XDebug документацией](#) для дополнительной информации.

## 2.51 Инструменты разработчика Phalcon (Developer Tools)

Эти инструменты представляют собой набор полезных скриптов для генерации основы приложения. Основные компоненты приложения могут быть созданы простыми командами, позволяющими легко разрабатывать приложения с использованием Phalcon.

**ВНИМАНИЕ:** Требуется Phalcon Framework версии 0.5.0 и выше. Так же рекомендуем PHP 5.3.6 и выше. если вы предпочитаете использовать веб-версию вместо консоли эта [блогозапись](#) содержит больше информации.

## 2.51.1 Скачать

Вы можете загрузить или скопировать инструменты разработчика с проекта на [Github](#), инструменты написаны на PHP и они кроссплатформенны.

### Установка

Существуют подробные инструкции о том, как установить средства разработки на различные платформы:

#### Инструменты разработчика Phalcon для Windows

Эти шаги помогут вам с установкой инструментов разработчика Phalcon на Windows.

**Предпосылки** Для запуска Инструментов разработчика необходимо установленное PHP расширение Phalcon. Если расширение еще не установлено, обратите внимание на инструкции в разделе [Установка](#).

**Скачать** Вы можете скачать кроссплатформенный пакет с инструментами разработчика в разделе [Скачать](#). Можно так же клонировать его с [Github](#).

На платформе Windows вам необходимо настроить системную переменную PATH для запуска инструментов разработчика и выполнения php. Если вы скачали инструменты разработчика в виде ZIP-архива, то его необходимо распаковать, например в *c:\phalcon-tools*. Запомните этот каталог, путь к нему понадобится ниже. Отредактируйте файл “phalcon.bat”, для этого кликните правой кнопкой мыши и выберите “Редактировать”:

Измените путь на тот, в который были установлены инструменты разработчика Phalcon:

Сохраните изменения.

**Добавление PHP и Инструментов в системную переменную PATH** Поскольку сценарии написаны на PHP, его необходимо установить на ваш компьютер. В зависимости от того, как был установлен PHP, его исполняемый файл может быть в разных местах. Найдите файл *php.exe* и запомните (скопируйте) путь к нему. Например, в последней редакции WAMP PHP по умолчанию располагается в: *C:\wamp\bin\php\php5.3.10\php.exe*.

В меню Пуск правой кнопкой кликните на значок “My Computer” и выберите “Properties”:

Выберите вкладку “Advanced” и нажмите кнопку “Environment Variables”: В Windows 7 (Компьютер - свойства - дополнительные параметры системы - параметры среды)

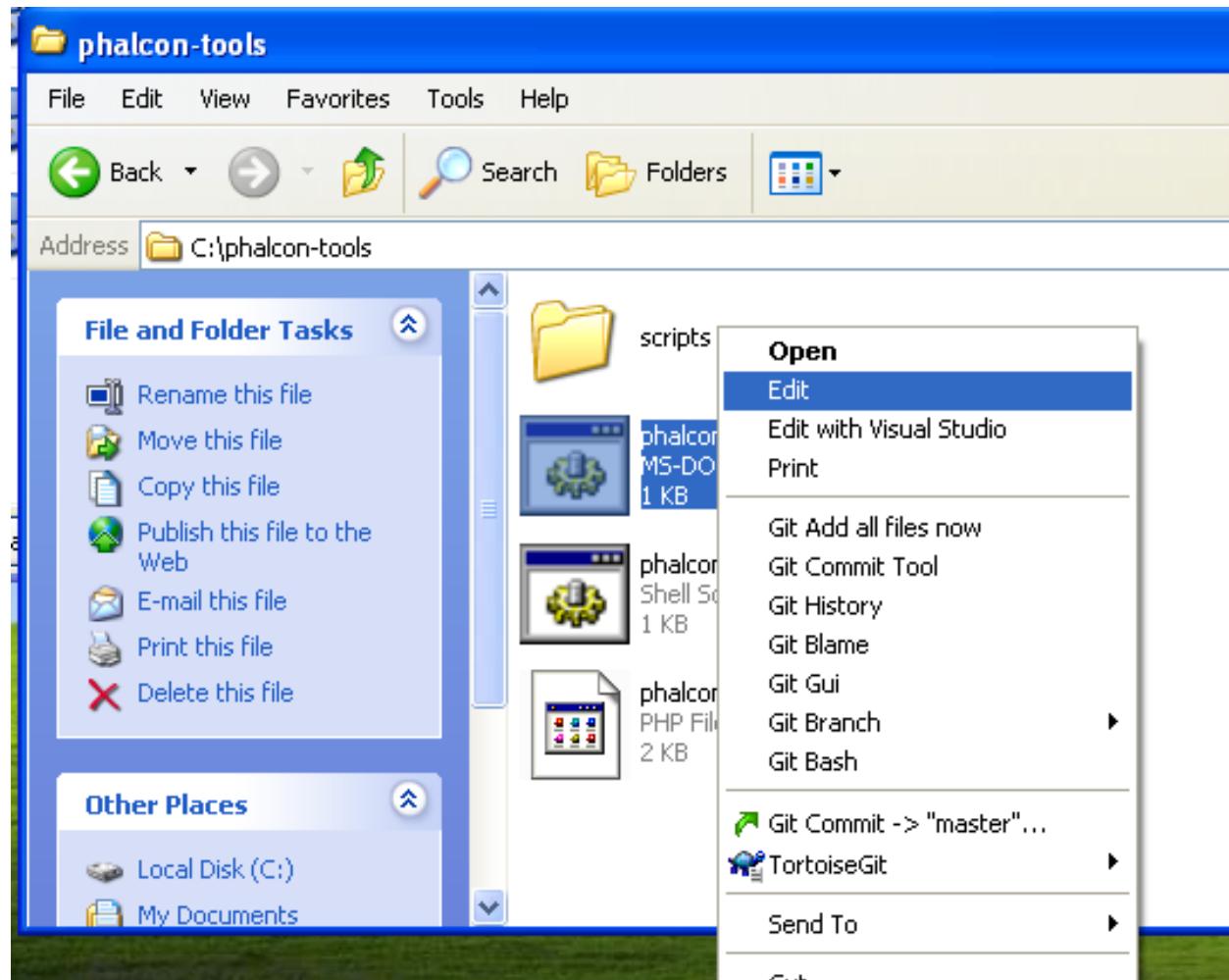
В нижней части диалога обратите внимание на раздел “System variables” и отредактируйте переменную “Path”:

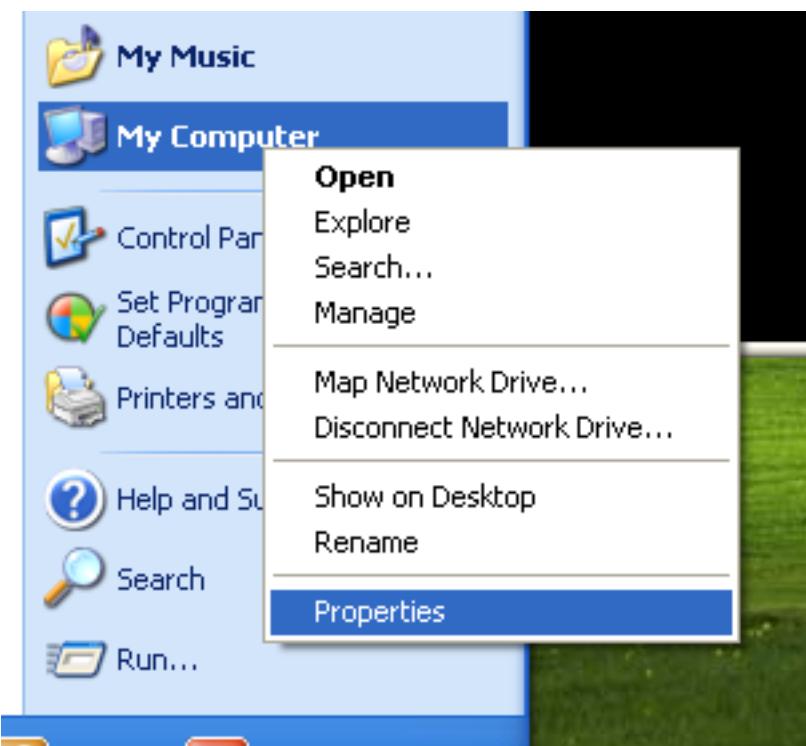
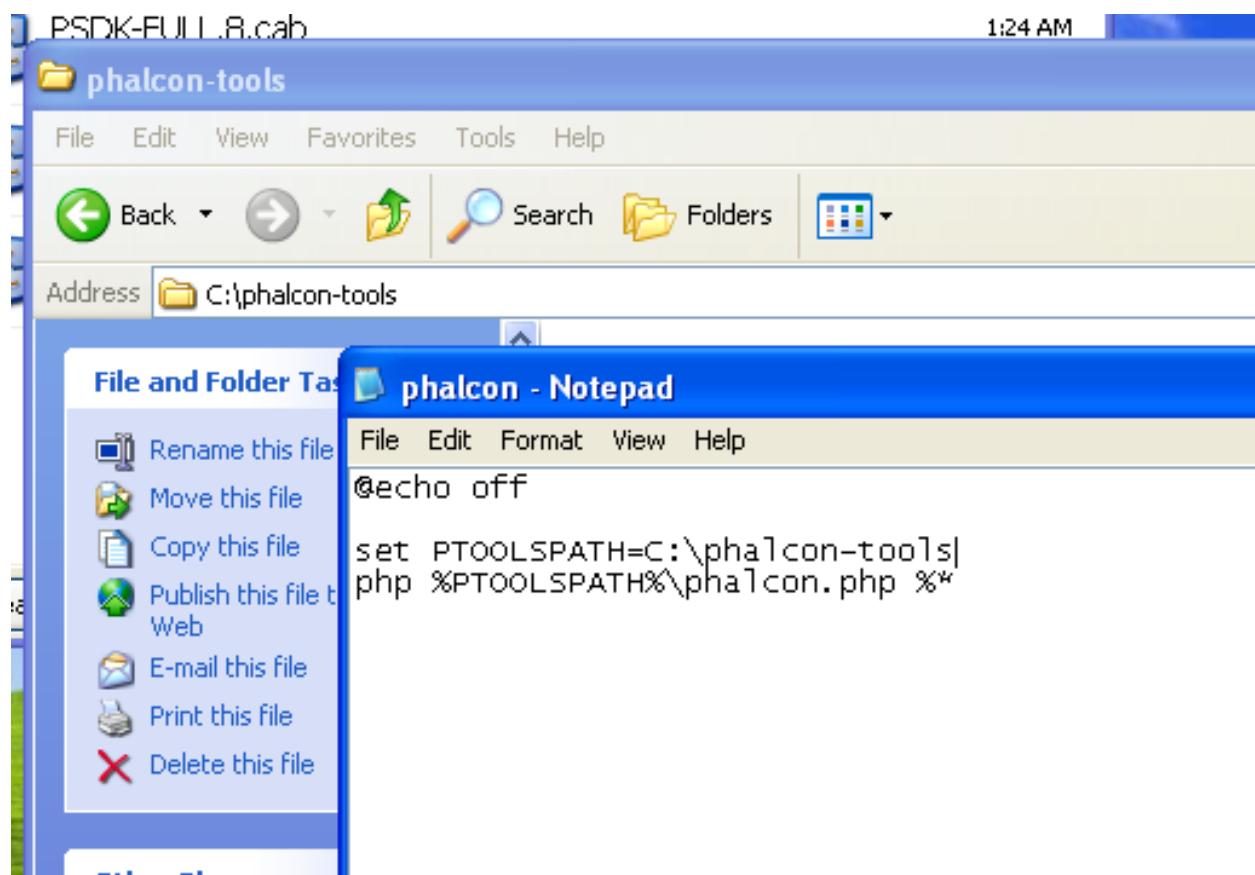
Будьте осторожны на этом шаге! В конце этой длинной строки вам надо будет добавить путь к установленному файлу *php.exe* и путь к установленным инструментам разработчика Phalcon. Используйте символ “;” для разделения:

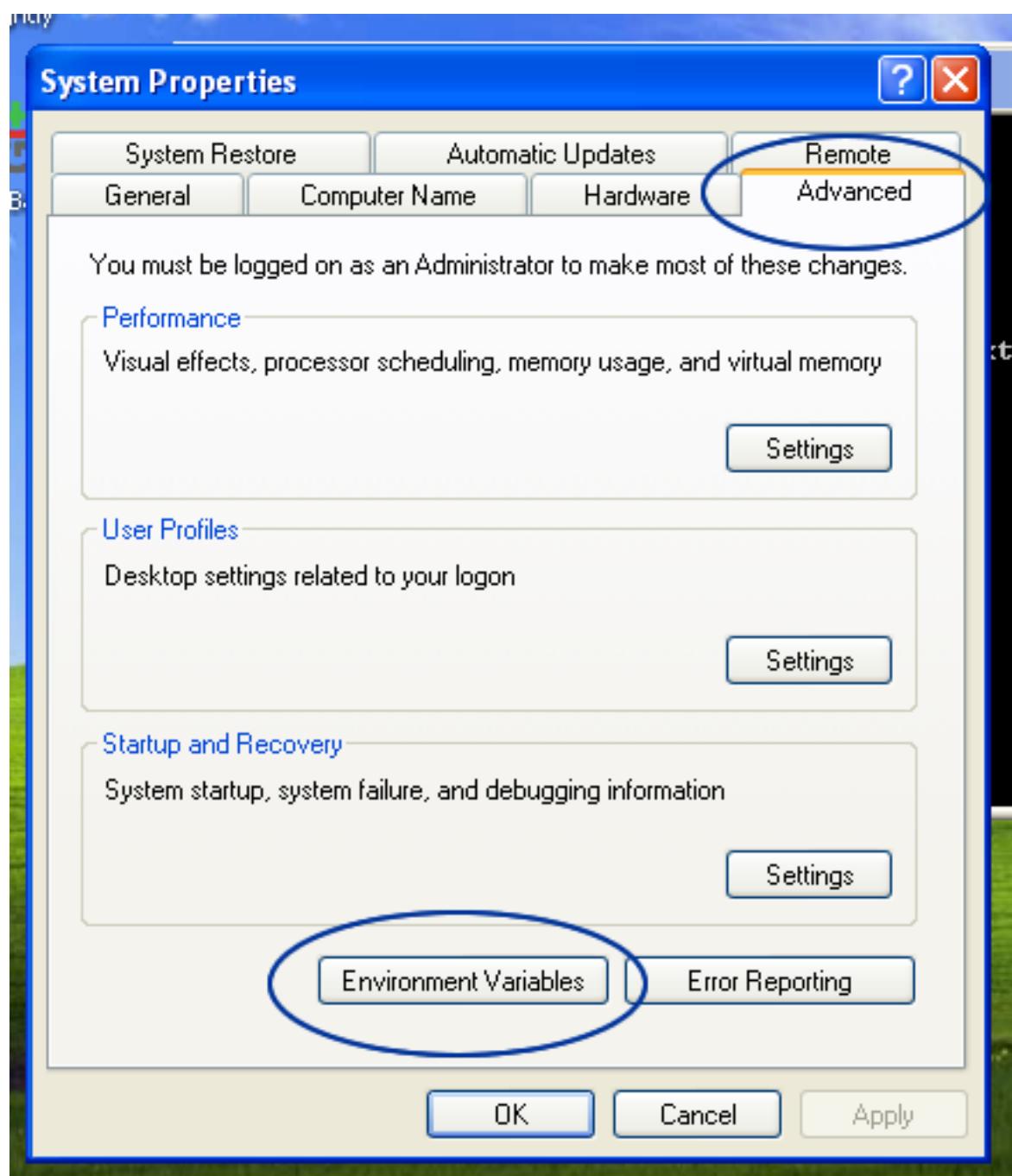
Примените изменения, нажав кнопку “Ok” и закройте диалог. В меню Пуск выберите поле “Run”. Если не можете найти эту опцию, нажмите сочетание “Windows” + “R”.

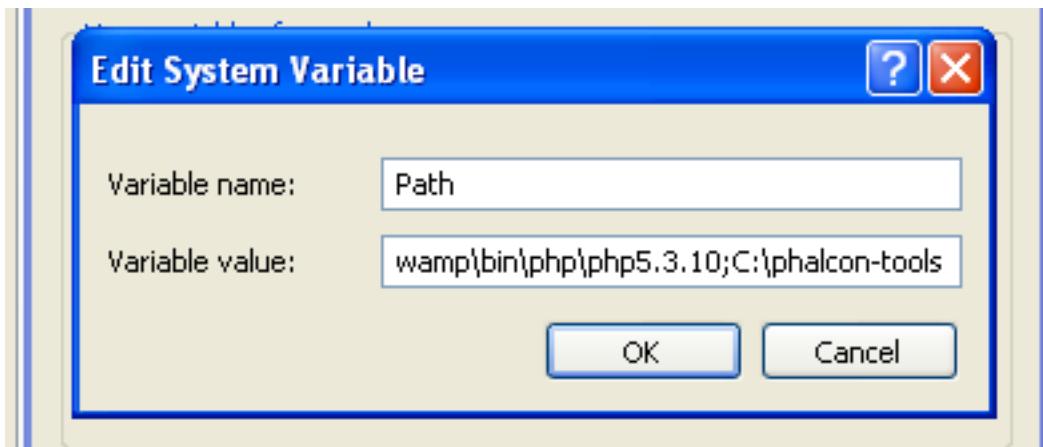
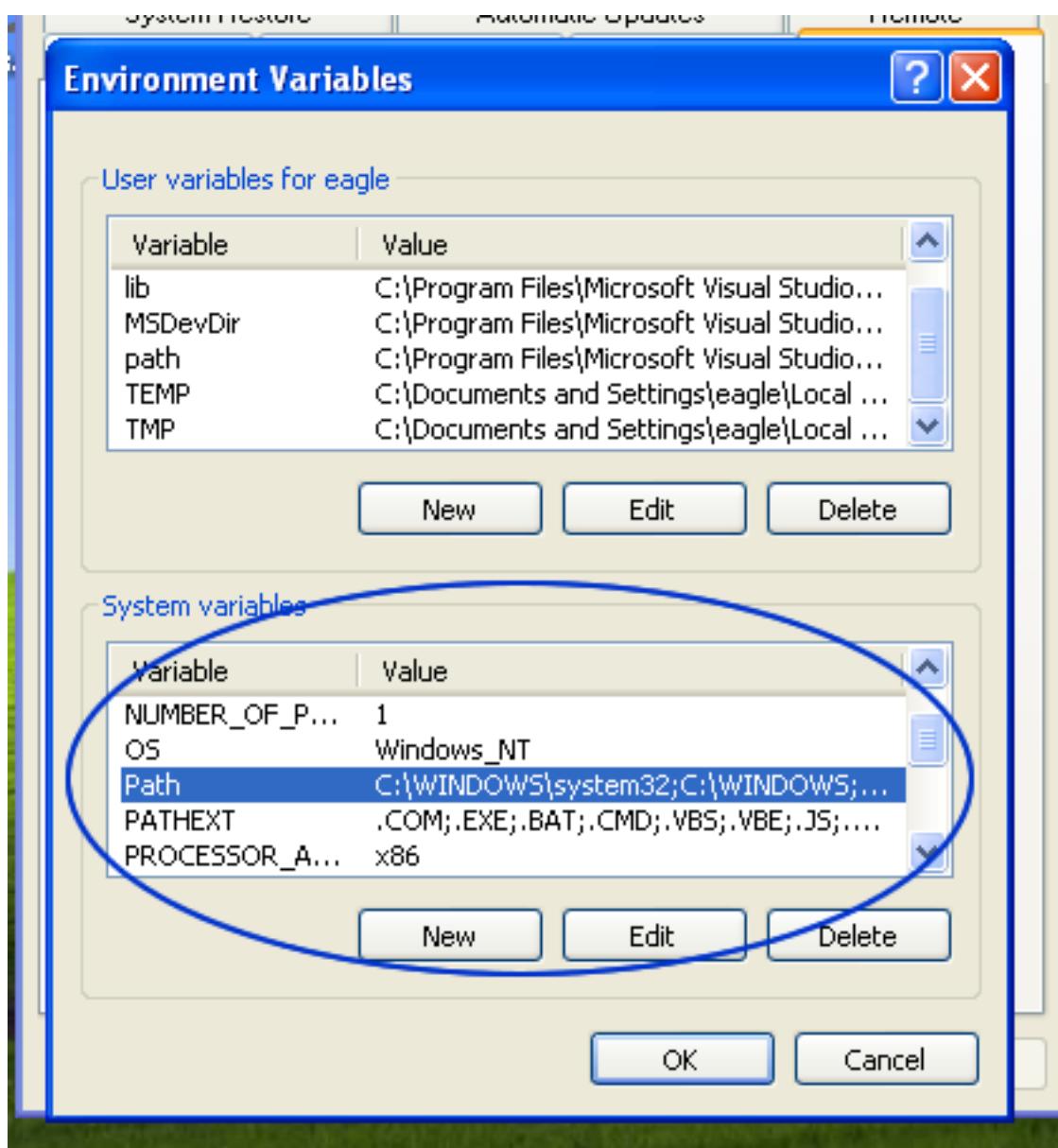
Выберите “cmd” и нажмите “Enter” для запуска окна консоли:

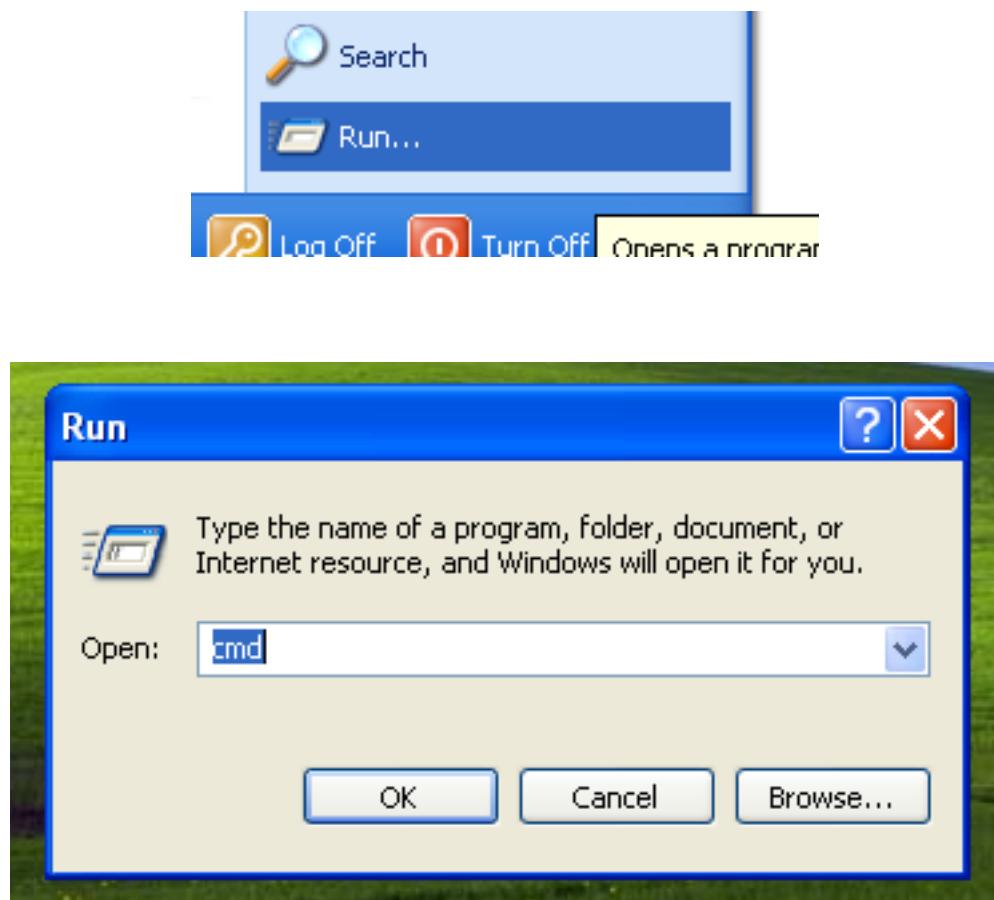
Введите команду “*php -v*” и “*phalcon*” и увидите что-то вроде этого:











```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\eagle>php -v
PHP 5.3.10 (cli) (built: Feb  2 2012 20:27:51)
Copyright (c) 1997-2012 The PHP Group
Zend Engine v2.3.0, Copyright (c) 1998-2012 Zend Technologies
    with Xdebug v2.1.2, Copyright (c) 2002-2011, by Derick Rethans

C:\Documents and Settings\eagle>phalcon
Phalcon: incorrect usage

C:\Documents and Settings\eagle>_
```

Поздравляем, инструменты разработчика Phalcon установлены!

### Дополнительные руководства

- [Использование инструментов разработчика](#)
- [Установка на OS X](#)
- [Установка на Linux](#)

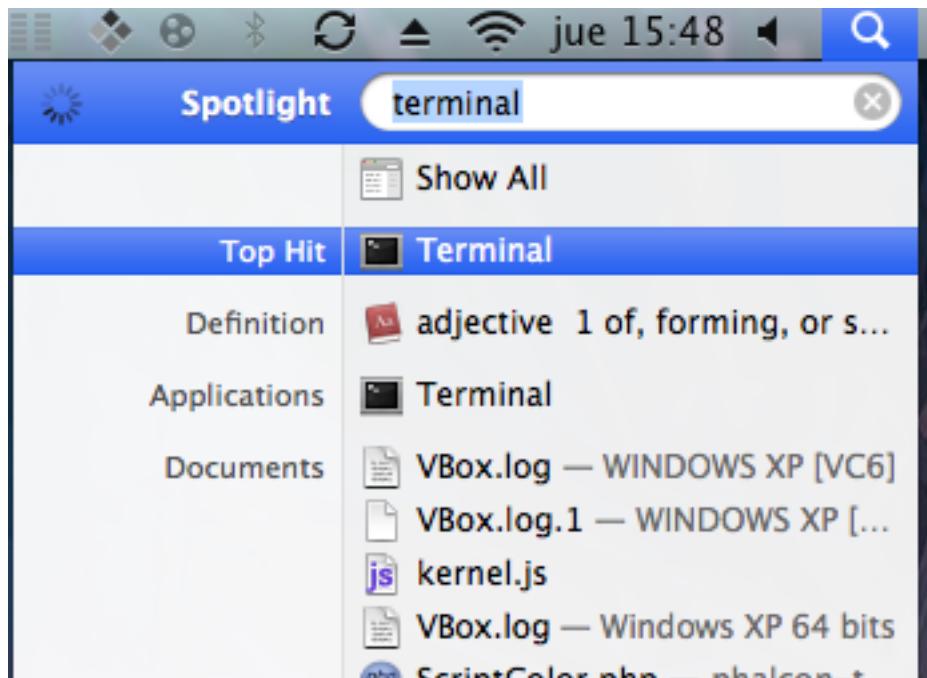
### Инструменты разработчика (Developer Tools) в Mac OS X

Эти действия покажут вам процесс установки Инструментов разработчика для OS/X.

**Предпосылки** Для запуска Инструментов разработчика необходимо установленное PHP расширение Phalcon. Если расширение еще не установлено, обратите внимание на инструкции в разделе [Установка](#).

**Скачать** Вы можете скачать кроссплатформенный пакет с инструментами разработчика в разделе [Скачать](#). Можно так же клонировать его с [Github](#).

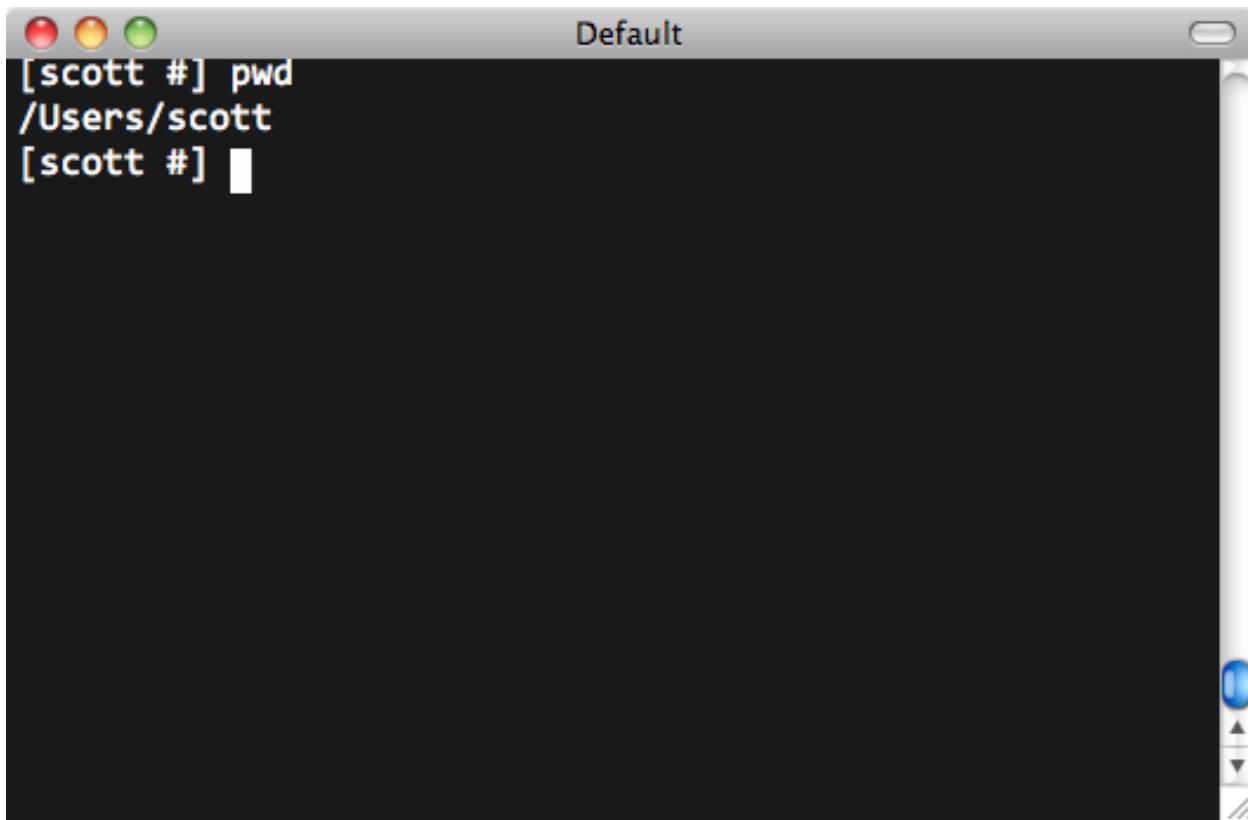
Откройте терминал:



Скопируйте и вставьте следующие команды в терминал:

```
wget -q --no-check-certificate -O phalcon-tools.zip http://github.com/phalcon/phalcon-devtools/zipball/master
unzip -q phalcon-tools.zip
mv phalcon-phalcon-devtools-* phalcon-tools
```

Проверьте, куда будут установлены Инструменты разработчика Phalcon, используя команду `pwd`:



На платформе Mac, вам необходимо самостоятельно настроить пользовательскую переменную PATH, для включения инструментов Phalcon. Откройте ваш .profile файл и добавьте путь к инструментам Phalcon в переменную PATH:

Вставьте две строки в конец файла:

```
export PATH=$PATH:/Users/scott/phalcon-tools  
export PTOOLSPATH=/Users/scott/phalcon-tools
```

Файл .profile должен быть примерно таким:

Сохраните изменения и закройте файл. Потом в терминале, введите следующие команды для создания символьической ссылки на скрипт phalcon.sh:

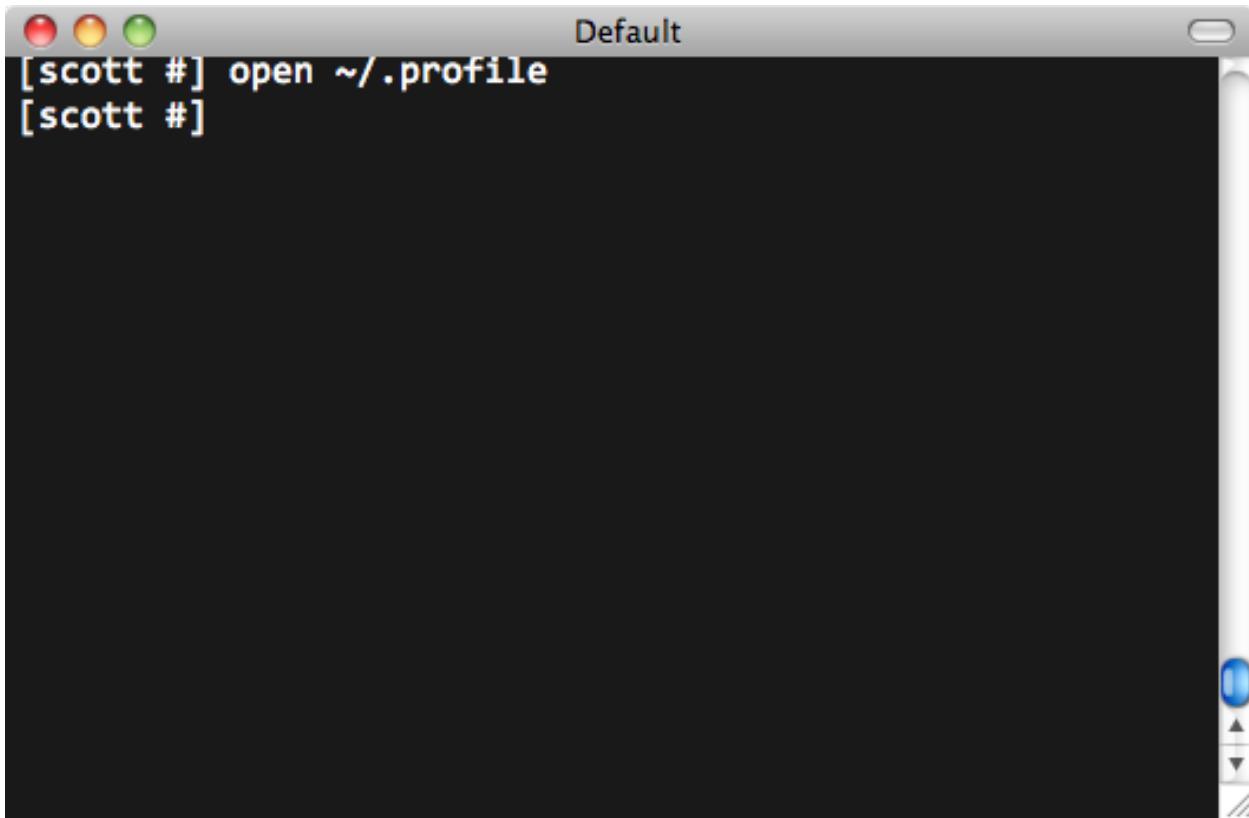
```
ln -s ~/phalcon-tools/phalcon.sh ~/phalcon-tools/phalcon  
chmod +x ~/phalcon-tools/phalcon
```

Введите команду “phalcon” и вы должны увидеть нечто подобное:

Поздравляем, инструменты разработчика Phalcon успешно установлены!

## Дополнительные руководства

- Использование инструментов разработчика (*Developer Tools*)
- Установка в *Windows*
- Установка в *Linux*



```
[scott #] open ~/.profile
[scott #]
```

## Инструменты разработчика Phalcon Developer Tools в Linux

Этот шаг поможет вам установить Phalcon Developer Tools в Linux.

**Требования** Для запуска инструментов разработчика необходимо установленное расширение Phalcon PHP. Если оно еще не установлено, прочитайте инструкции в разделе [Installation](#).

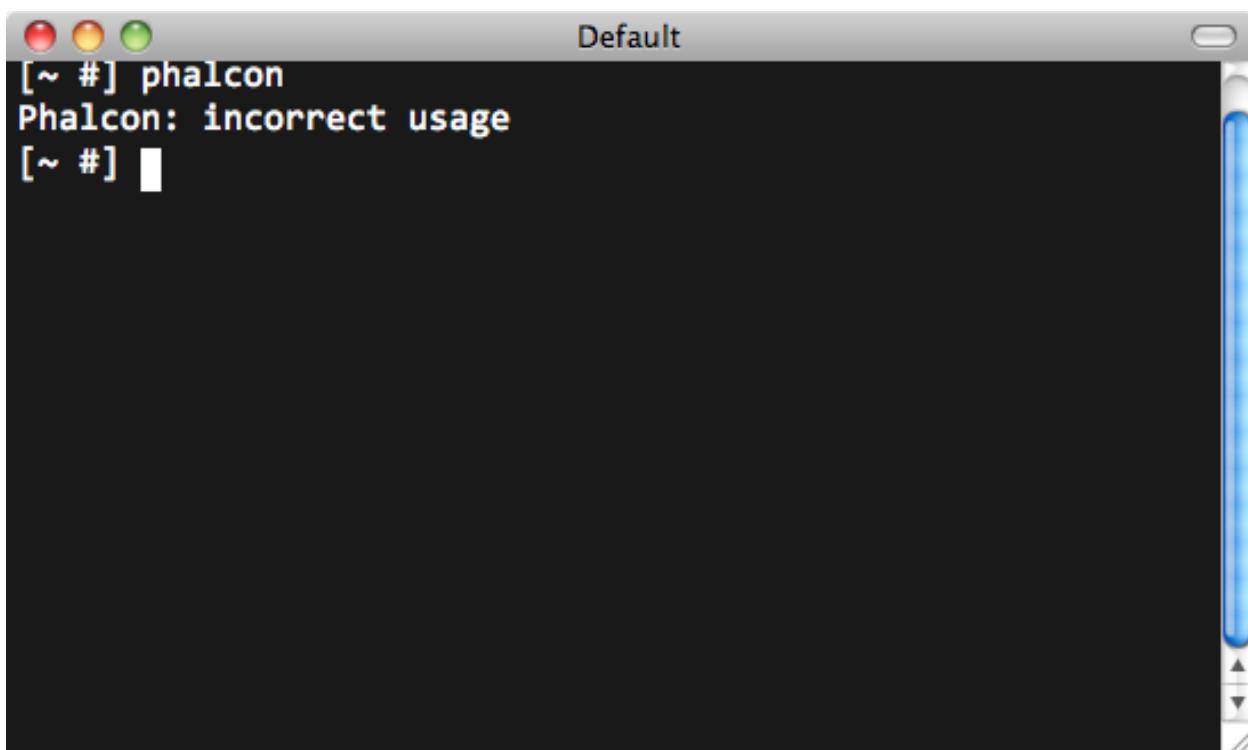
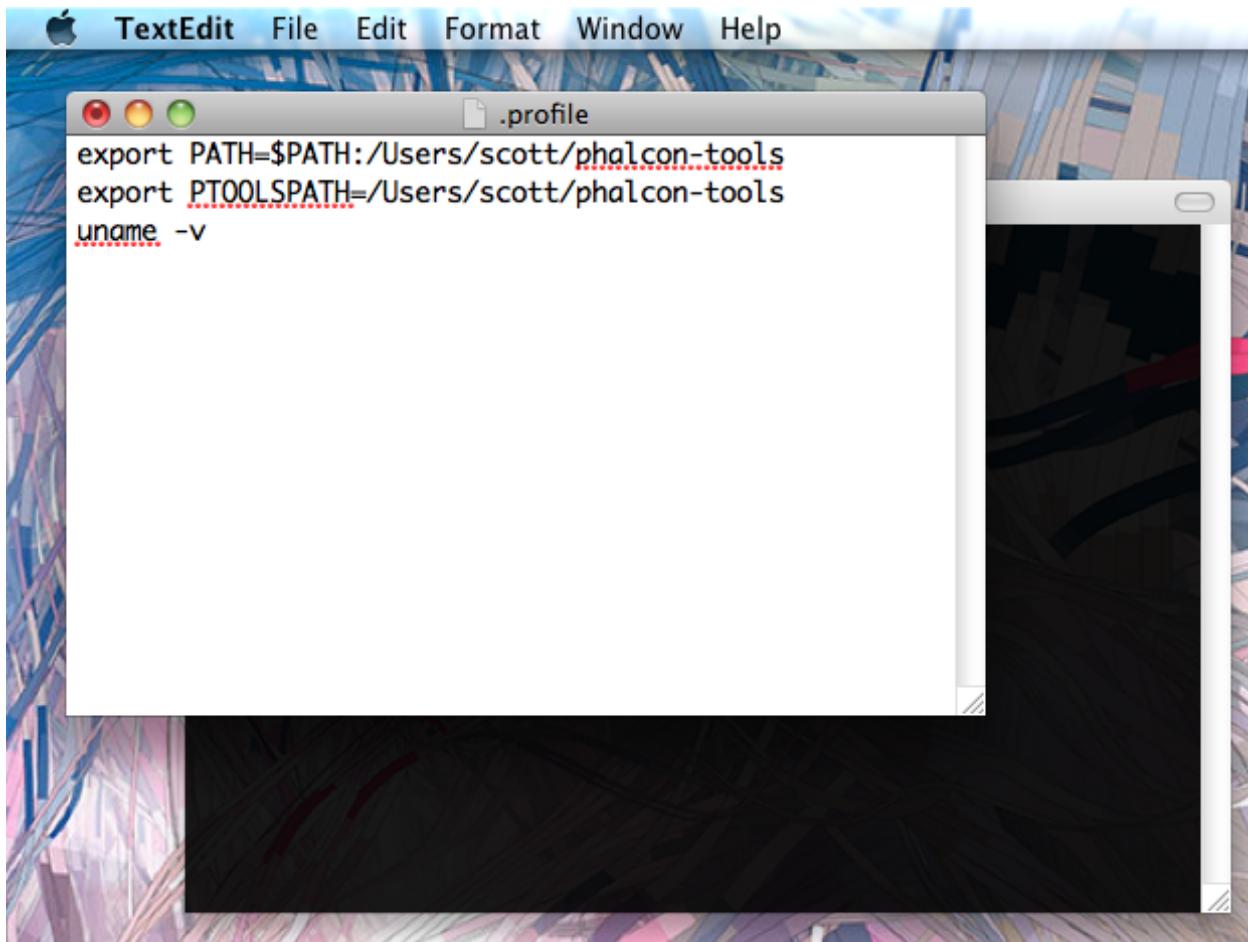
**Скачать** Вы можете скачать кроссплатформенный пакет инструментов разработчиков из раздела [Download](#). Также вы можете его слить (`git clone`) на [Github](#). Введите в терминале следующие команды:

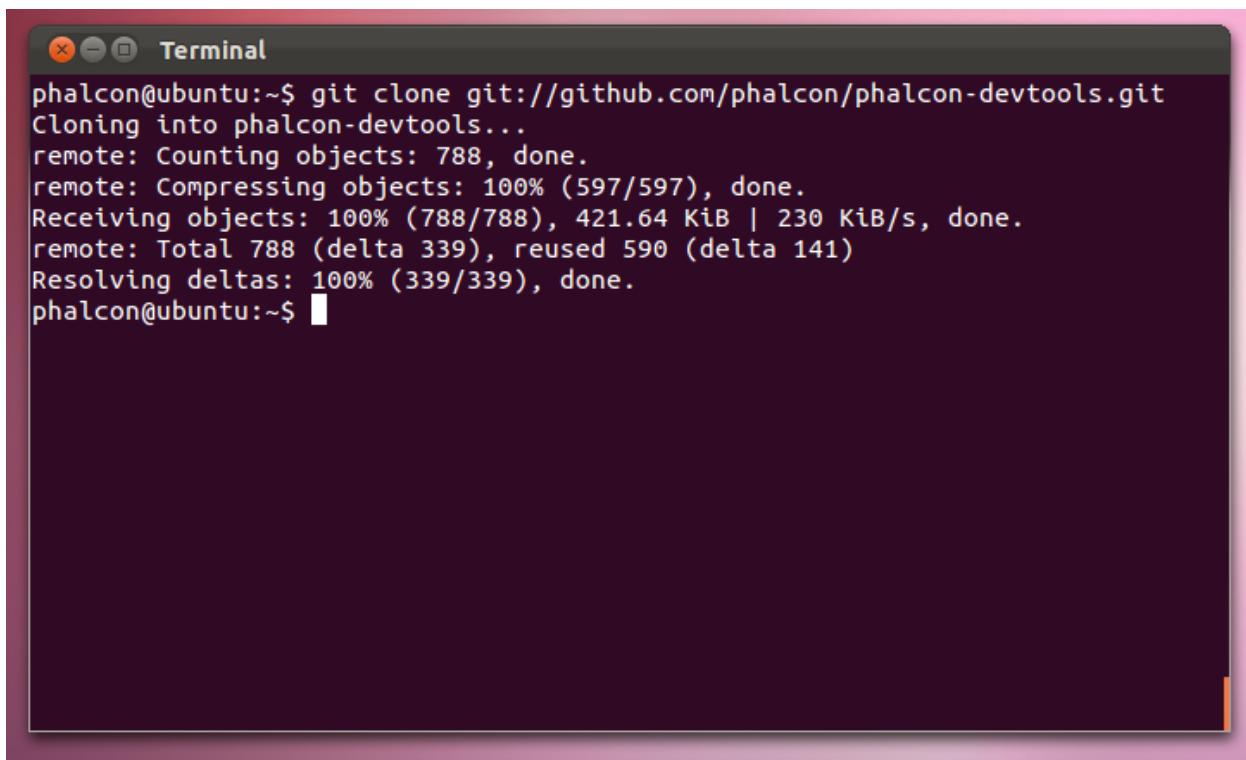
Затем откройте папку, в которую были скопированы инструменты, и выполните команду ”`./phalcon.sh`”, (не забудьте точку в начале команды):

Поздравляем, инструменты разработчика Phalcon установлены!

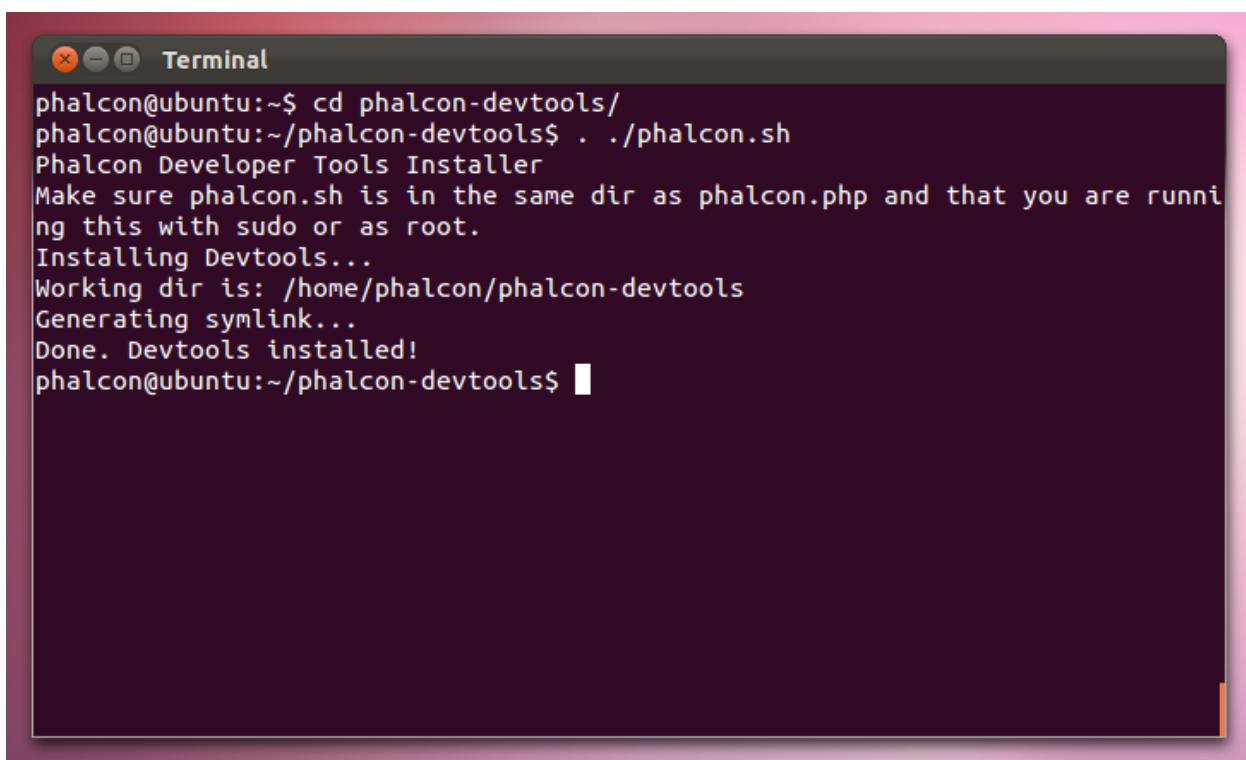
## Дополнительные руководства

- [Использование инструментов разработчика](#)
- [Установка на Windows](#)
- [Установка на Mac](#)





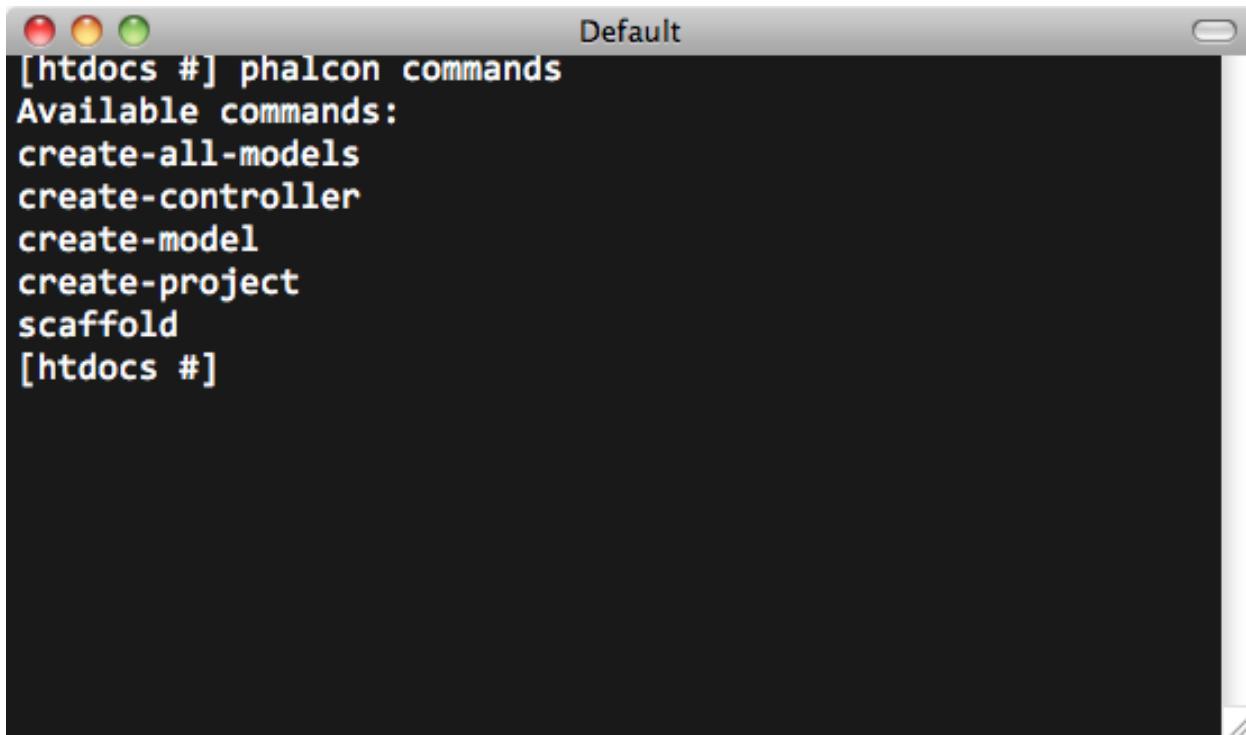
```
phalcon@ubuntu:~$ git clone git://github.com/phalcon/phalcon-devtools.git
Cloning into phalcon-devtools...
remote: Counting objects: 788, done.
remote: Compressing objects: 100% (597/597), done.
Receiving objects: 100% (788/788), 421.64 KiB | 230 KiB/s, done.
remote: Total 788 (delta 339), reused 590 (delta 141)
Resolving deltas: 100% (339/339), done.
phalcon@ubuntu:~$
```



```
phalcon@ubuntu:~$ cd phalcon-devtools/
phalcon@ubuntu:~/phalcon-devtools$ ./phalcon.sh
Phalcon Developer Tools Installer
Make sure phalcon.sh is in the same dir as phalcon.php and that you are running this with sudo or as root.
Installing Devtools...
Working dir is: /home/phalcon/phalcon-devtools
Generating symlink...
Done. Devtools installed!
phalcon@ubuntu:~/phalcon-devtools$
```

### 2.51.2 Получение списка команд

Для получения списка имеющихся команд введите: phalcon commands



```
[htdocs #] phalcon commands
Available commands:
create-all-models
create-controller
create-model
create-project
scaffold
[htdocs #]
```

### 2.51.3 Создание скелета проекта

Вы можете использовать инструменты для создания скелета проекта на Phalcon. По умолчанию созданный проект будет использовать mod\_rewrite для Apache. Введите следующие команды в корне сайта вашего веб-сервера:

Проект создастся с полной рекомендованной структурой:

Для получения подробной информации по командам стоит использовать параметр *-help*:

Созданный проект можно сразу запустить в браузере:

### 2.51.4 Создание контроллеров

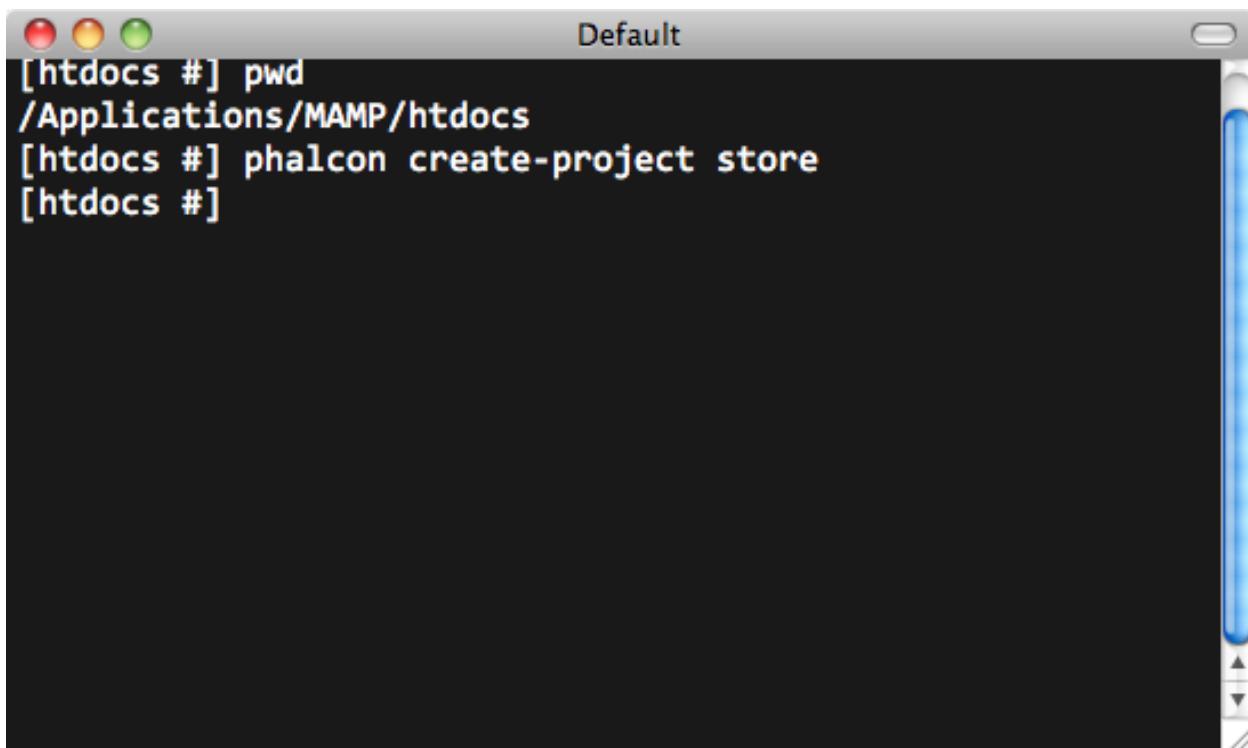
Команда “create-controller” генерирует заготовку контроллера. Её необходимо выполнять в корне существующего проекта Phalcon:

Команда выше сформирует следующий код:

```
<?php

class TestController extends Phalcon\Mvc\Controller
{

    public function indexAction()
    {
```



```
[htdocs #] pwd  
/Applications/MAMP/htdocs  
[htdocs #] phalcon create-project store  
[htdocs #]
```

```
}
```

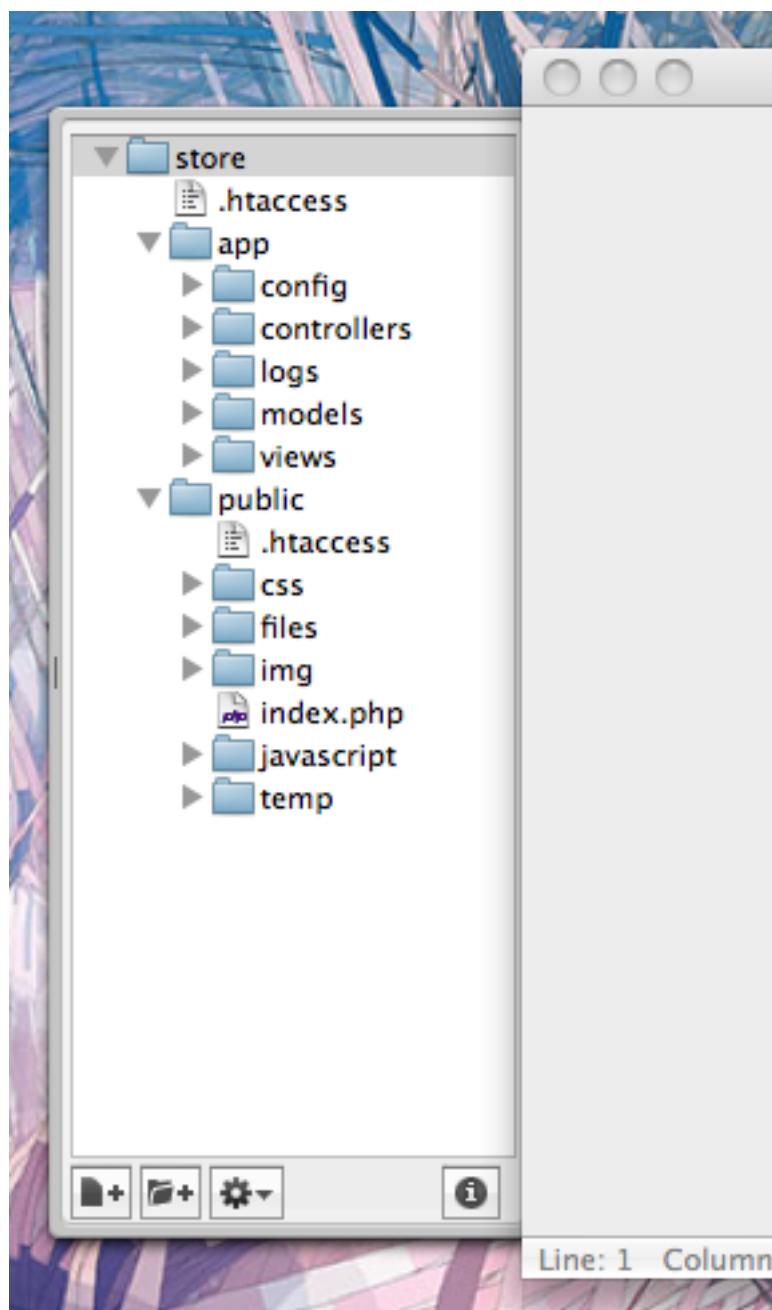
```
}
```

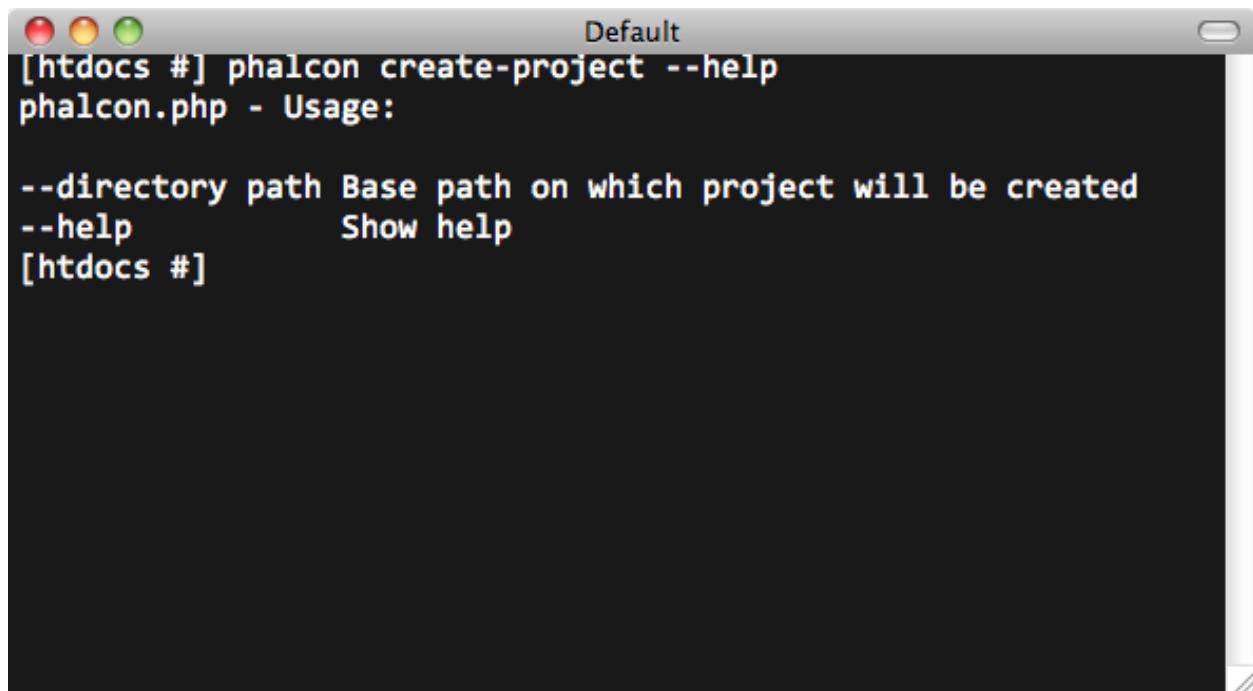
### 2.51.5 Настройка параметров базы данных

В проектах, созданных с использованием средств разработчика файл конфигурации будет искааться в *app/config/config.ini* или *app/config/config.php*. Для создания моделей или каркасов(scaffold), вам потребуется изменить настройки, используемые для подключения к вашей базе данных.

При использовании конфигурации в файле config.ini:

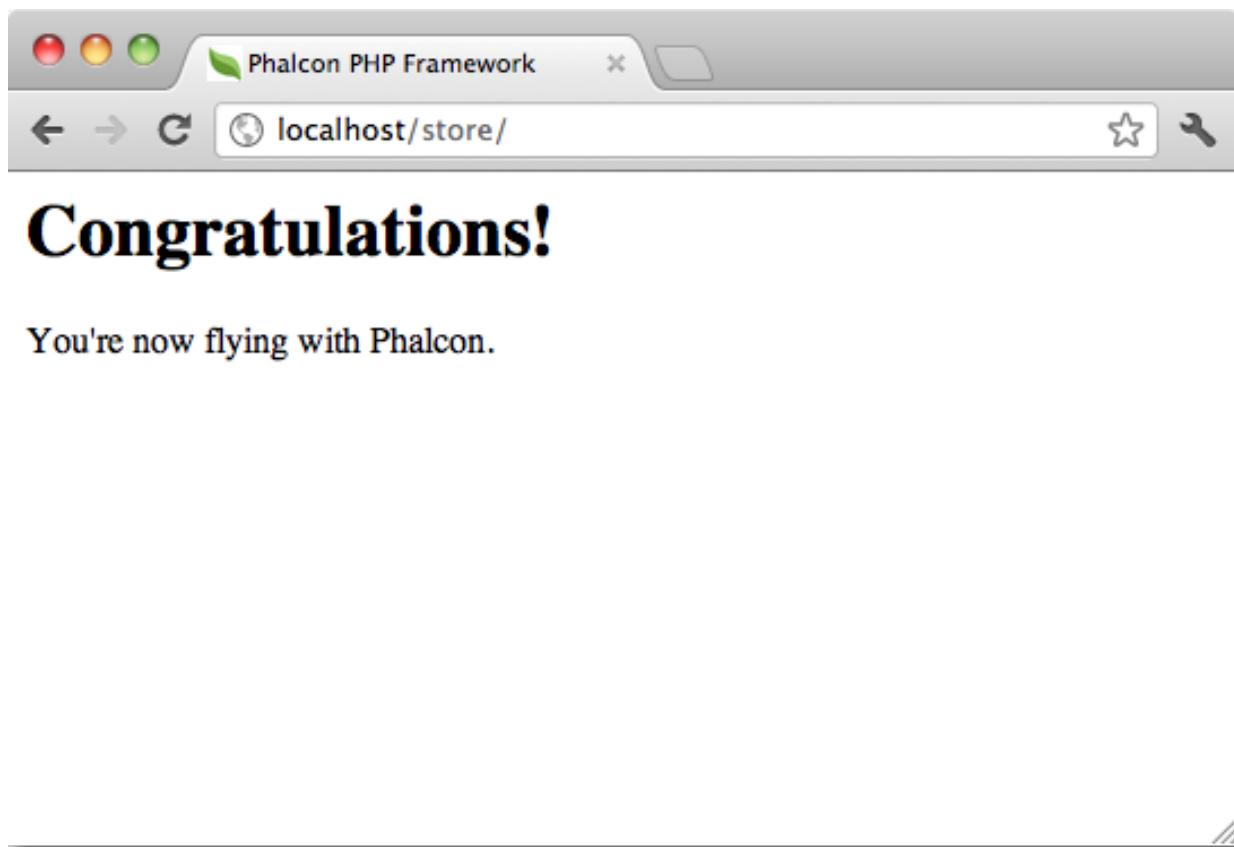
```
[database]  
adapter = Mysql  
host = "127.0.0.1"  
username = "root"  
password = "secret"  
dbname = "store_db"  
  
[phalcon]  
controllersDir = "../app/controllers/"  
modelsDir = "../app/models/"  
viewsDir = "../app/views/"  
baseUri = "/store/"
```

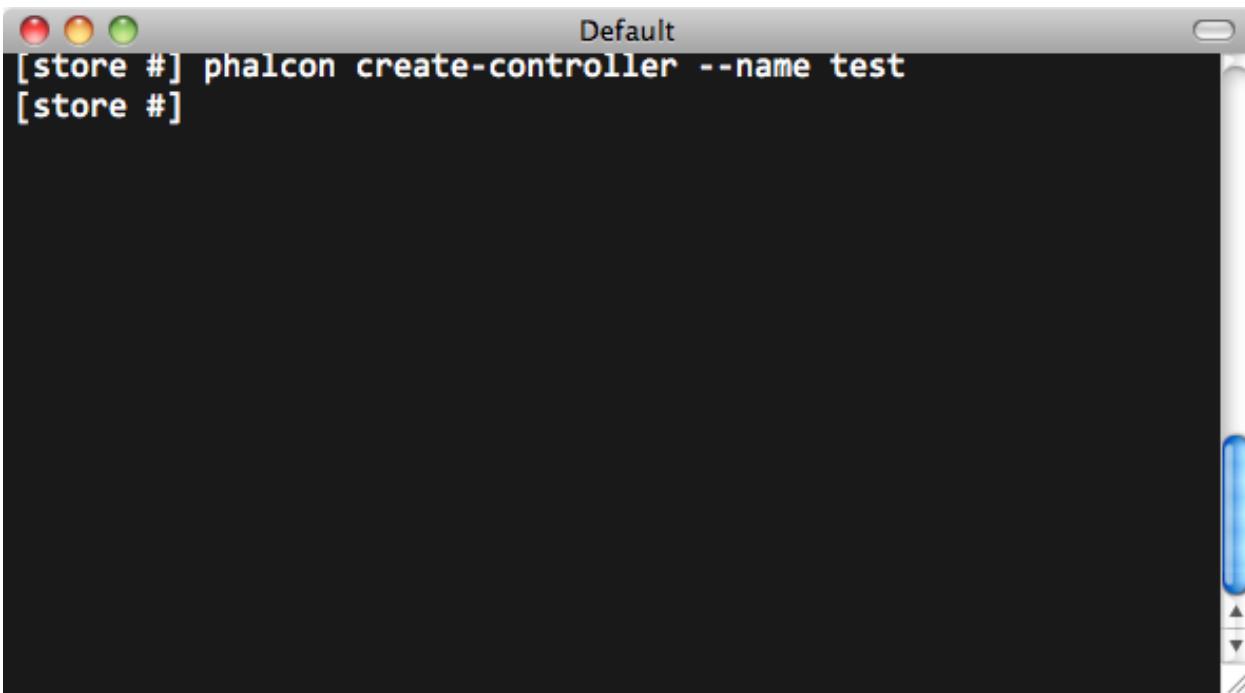




```
Default
[htdocs #] phalcon create-project --help
phalcon.php - Usage:

--directory path Base path on which project will be created
--help           Show help
[htdocs #]
```





```
[store #] phalcon create-controller --name test
[store #]
```

### 2.51.6 Создание моделей

Существует несколько способов генерации моделей. Вы можете создать все модели по таблицам текущей базы данных или для любой таблицы выборочно. Модели может содержать публичные атрибуты или работу через сеттеры (set{Field}) и геттеры (get{Field}). Самый простой способ для создания модели:

Созданная модель содержит публичные атрибуты для прямого доступа.

```
<?php
```

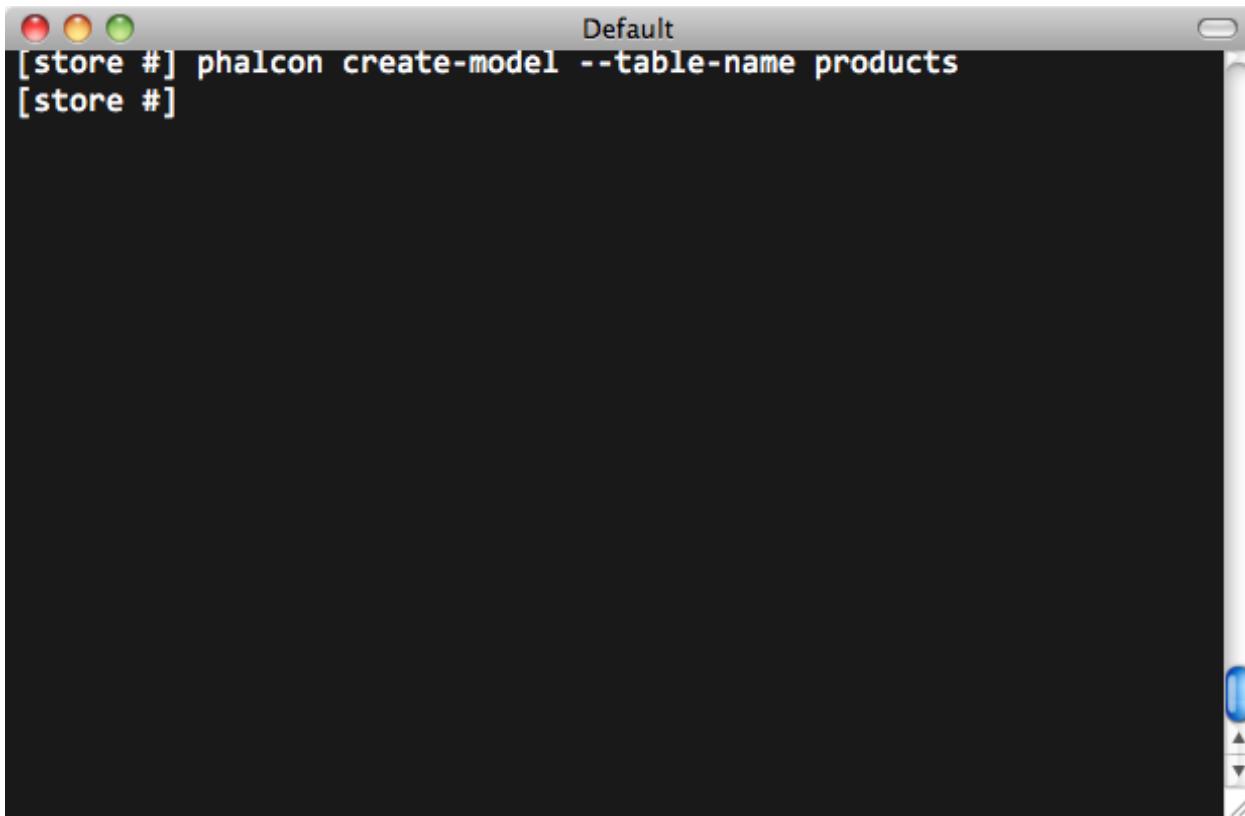
```
class Products extends \Phalcon\Mvc\Model
{

    /**
     * @var integer
     */
    public $id;

    /**
     * @var integer
     */
    public $types_id;

    /**
     * @var string
     */
    public $name;

    /**
     * @var string
     */
    public $price;
```



```
[store #] phalcon create-model --table-name products
[store #]
```

```
/**
 * @var integer
 */
public $quantity;

/**
 * @var string
 */
public $status;

}
```

При использовании `-get-set` атрибуты модели будут закрыты для прямого изменения, работа с ними будет только через соответствующие `get` и `set` методы. Такое поведение позволит изменить бизнес-логику работы модели внутри соответствующих `set/get` методов.

```
<?php

class Products extends \Phalcon\Mvc\Model
{

    /**
     * @var integer
     */
    protected $id;

    /**
     * @var integer
     */

    /**
     * @var integer
     */
```

```
/*
protected $types_id;

/**
 * @var string
 */
protected $name;

/**
 * @var string
 */
protected $price;

/**
 * @var integer
 */
protected $quantity;

/**
 * @var string
 */
protected $status;

/**
 * Метод установки значения для поля id
 * @param integer $id
 */
public function setId($id)
{
    $this->id = $id;
}

/**
 * Метод установки значения для поля types_id
 * @param integer $types_id
 */
public function setTypesId($types_id)
{
    $this->types_id = $types_id;
}

...

/**
 * Возвращаем значение статуса поля
 * @return string
 */
public function getStatus()
{
    return $this->status;
}

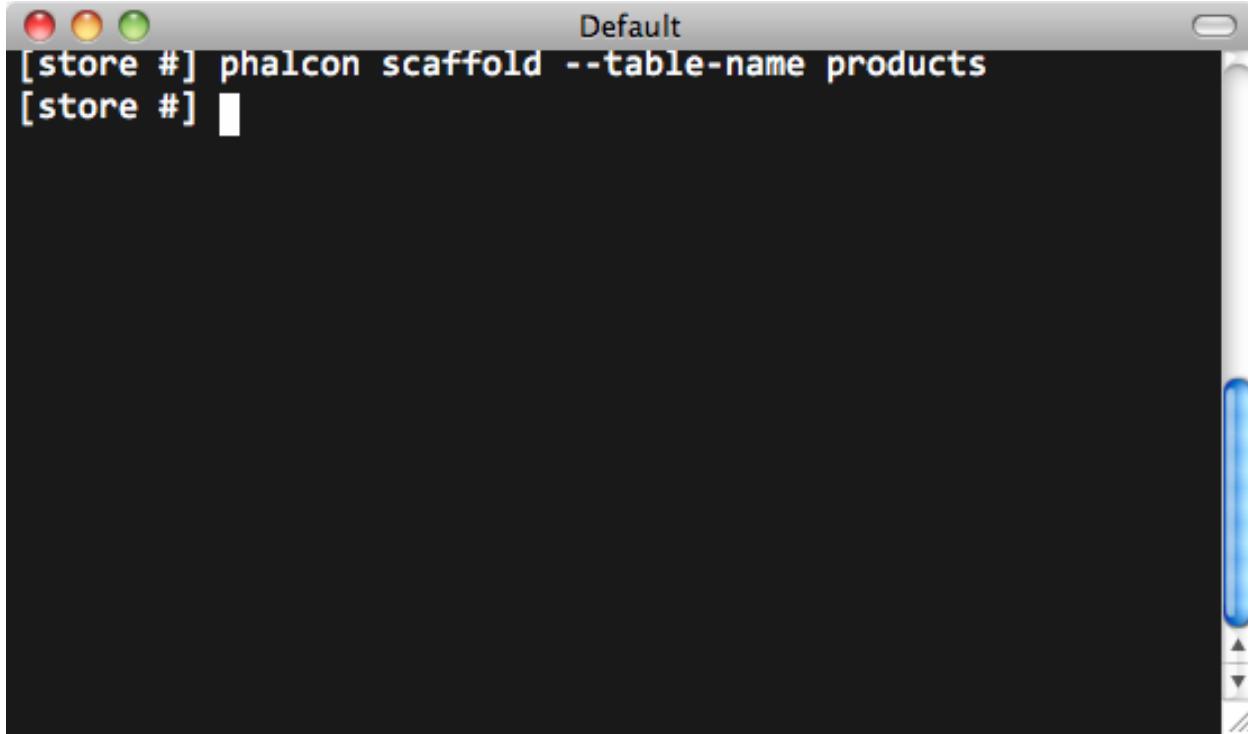
}
```

Приятной особенностью генератора моделей является то, что он сохраняет изменения, сделанные разработчиком. Это позволяет добавлять или удалять поля и свойства, не беспокоясь о потере изменений, внесенных в модель вручную. Следующий демо-ролик показывает как это работает:

## 2.51.7 Автоматическая генерация CRUD интерфейса (Scaffolding)

Scaffolding - это быстрый способ для получения основных элементов приложения. Если вы хотите быстро создать модели, представления, и контроллеры для нового ресурса приложения - использование автоматической генерации кода является отличным инструментом для этих задач.

После того, как код сгенерирован, его можно настроить под себя. Многие разработчики не используют scaffolding, предпочитая писать весь код самостоятельно. Сгенерированный код может служить в качестве руководства, чтобы лучше понять основы работы или разработки прототипов. Видео ниже показывает генерацию интерфейса для таблицы “товары (products)”:



```
Default
[store #] phalcon scaffold --table-name products
[store #]
```

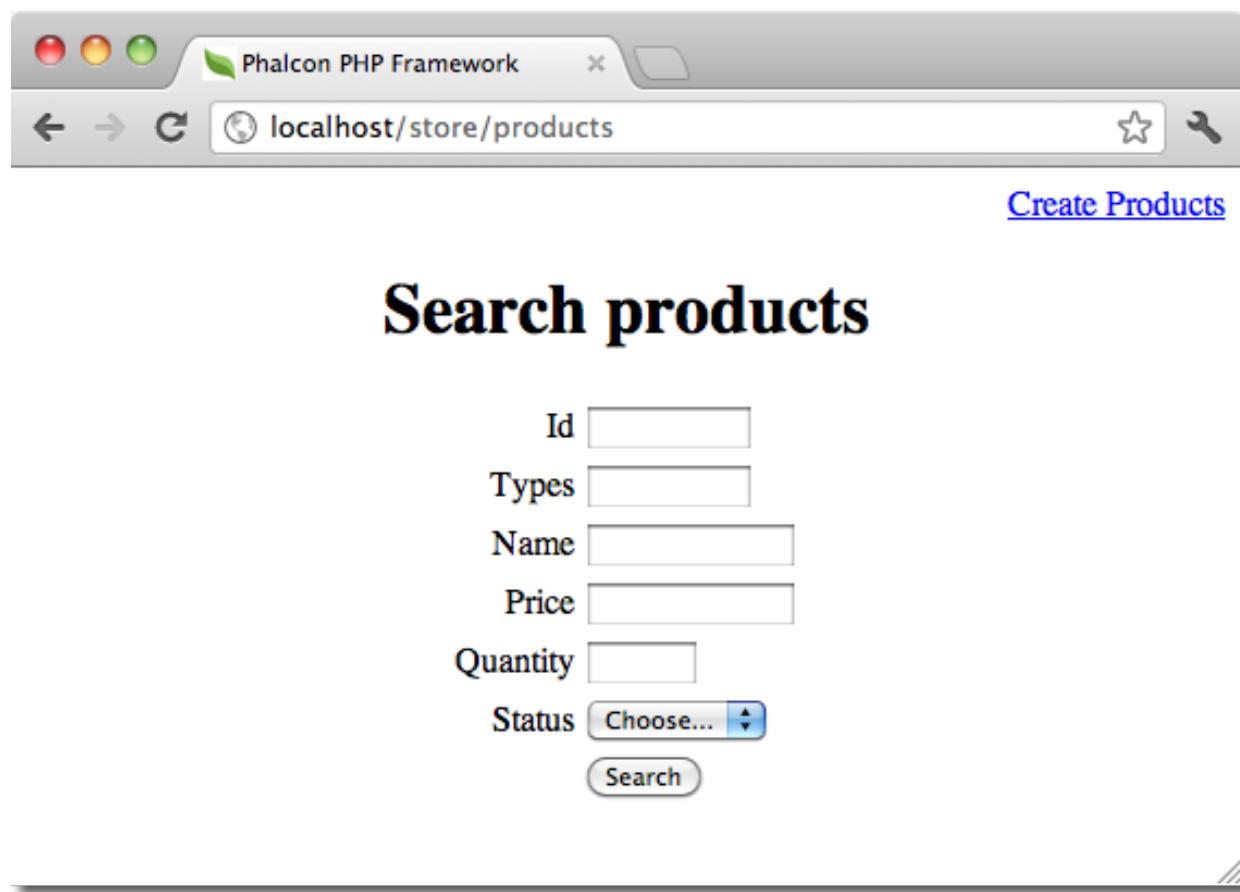
Генератор создаст несколько файлов в вашем приложении, и каталоги для них. Вот краткий обзор того, что будет сгенерировано:

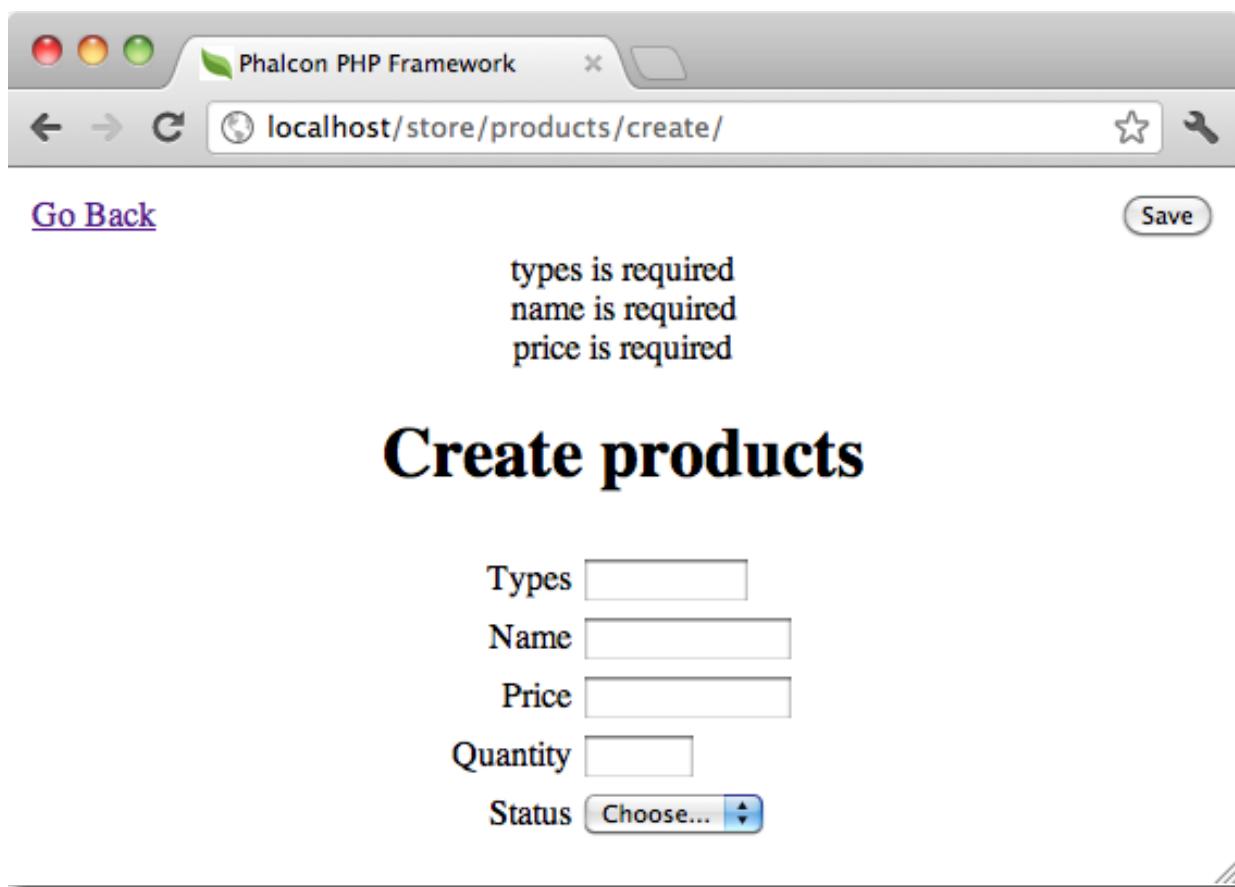
| Файл                                   | Предназначение                      |
|--|-------------------------------------|
| app/controllers/ProductsController.php | Контроллер продуктов                |
| app/models/Products.php                | Модель продуктов                    |
| app/views/layout/products.phtml        | Макет для контроллера продуктов     |
| app/views/products/new.phtml           | Представление для действия “new”    |
| app/views/products/edit.phtml          | Представление для действия “edit”   |
| app/views/products/search.phtml        | Представление для действия “search” |
| app/views/products/edit.phtml          | Представление для действия “edit”   |

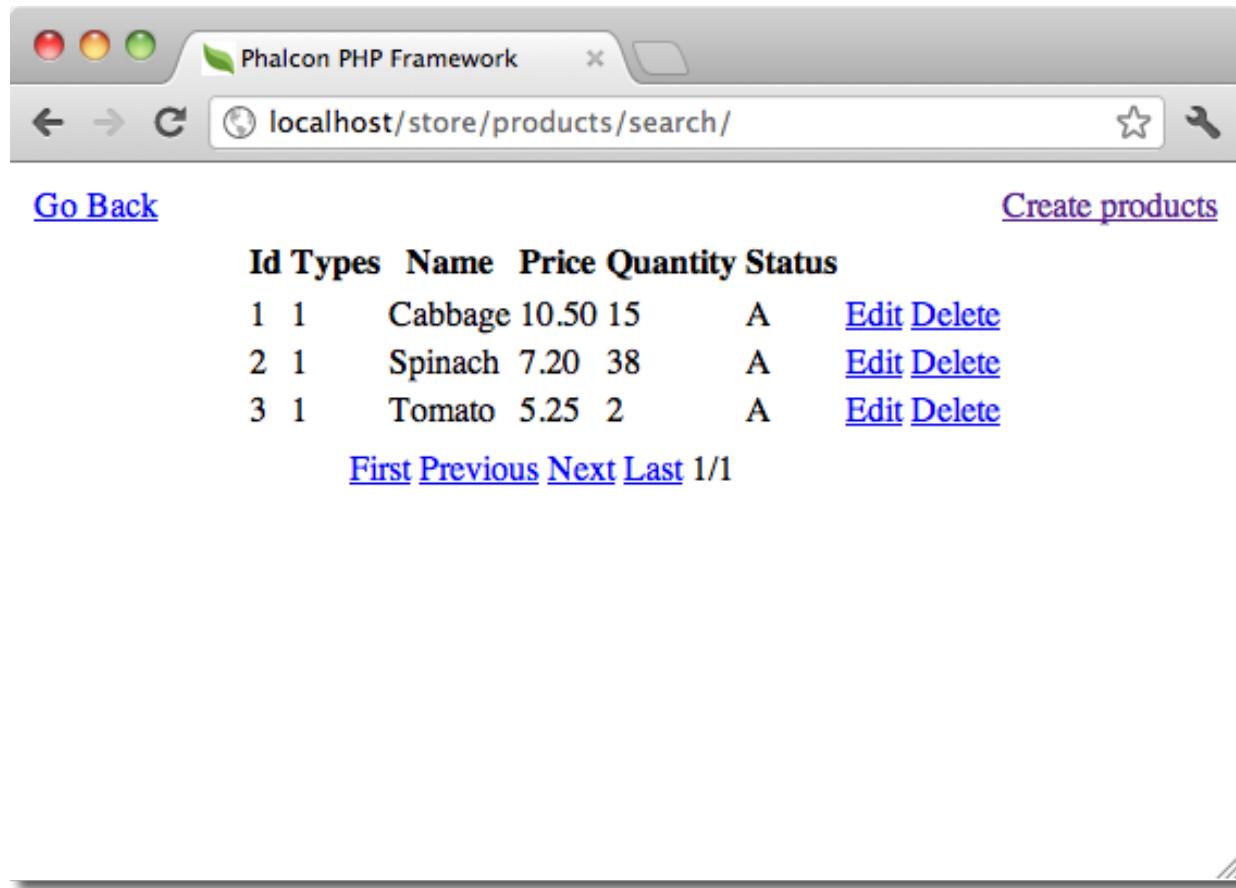
На главной странице созданного таким образом контроллеров вы увидите форму поиска, и ссылку на создание нового Продукта:

Страница создания товара позволяет добавить в таблицу products новую запись, при этом будут использованы проверки по правилам модели Products. Phalcon будет автоматически проверять “not null” поля и выдавать требования о их заполнении.

После выполнения поиска будут выведены результаты с постраничной навигацией. Используйте ссылки меню “Правка” или “Удалить” возле каждого результата поиска, для выполнения требуемых действий.







### 2.51.8 Веб интерфейс дополнений ( webtools )

Кроме того, можно использовать инструменты разработчика Phalcon через веб-интерфейс. Подробности его работы показаны на следующем скринкасте:

### 2.51.9 Интеграция в PhpStorm IDE

Скринкаст показывает, как интегрировать инструменты для разработчиков с PhpStorm IDE. Аналогично можно интегрировать дополнения в любой другой PHP редактор или IDE.

### 2.51.10 Заключение

Phalcon Developer Tools позволяет легко генерировать код для вашего приложения, сокращая время разработки и потенциальные ошибки.

## 2.52 Повышение производительности: Что дальше?

Для получения более быстрого приложения требуется уточнение множества аспектов: сервера, клиента, сети, базы данных, веб-сервера, статических источников и т.д. Здесь мы рассмотрим сценарии по улучшению производительности и поиску наиболее медленных мест приложения.

### 2.52.1 Профилирование на сервере

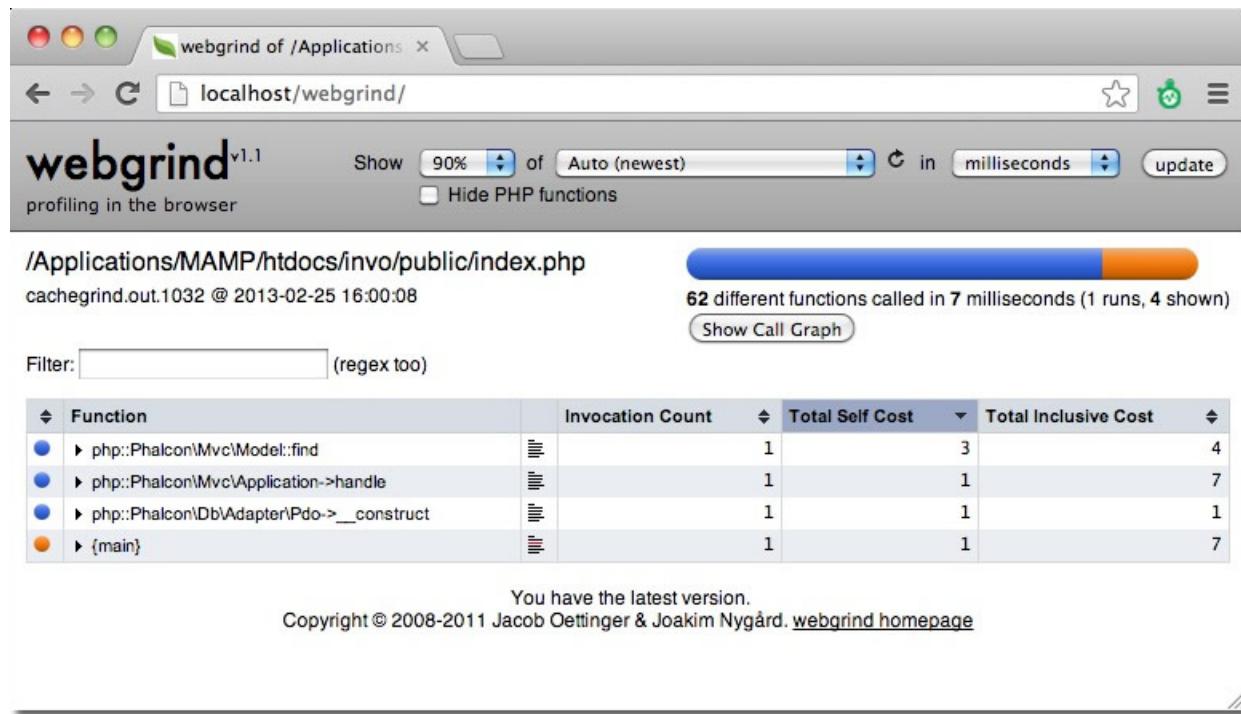
Все приложения разные, использование постоянного профилирования даёт понять, где можно увеличить производительность. Профилирование дает нам реальное представление о наиболее медленных местах. Результаты замеров могут различаться от раза к разу, поэтому необходимо сделать достаточное количество измерений, чтобы сделать правильные выводы.

#### Профилирование с XDebug

Xdebug предоставляет простой способ профилирования приложений на PHP. Просто установите расширение и включите его в php.ini:

```
xdebug.profiler_enable = On
```

С помощью инструмента `Webgrind` можно визуально понять, какие функции и методы медленнее остальных:



#### Профилирование с Xhprof

Xhprof — еще одно интересное расширение для профилирования PHP-приложений. Добавьте следующую строку в начало загрузочного файла:

```
<?php
```

```
xhprof_enable(XHPROF_FLAGS_CPU + XHPROF_FLAGS_MEMORY);
```

Потом, в конце файла, сохраните данные профилирования:

```
<?php
```

```
$xhprof_data = xhprof_disable('/tmp');

$XHPROF_ROOT = "/var/www/xhprof/";
include_once $XHPROF_ROOT . "/xhprof_lib/utils/xhprof_lib.php";
include_once $XHPROF_ROOT . "/xhprof_lib/utils/xhprof_runs.php";

$xhprof_runs = new XHProfRuns_Default();
$run_id = $xhprof_runs->save_run($xhprof_data, "xhprof_testing");

echo "http://localhost/xhprof/xhprof_html/index.php?run={$run_id}&source=xhprof_testing\n";
```

Xhprof обеспечивает встроенный просмотрщик для анализа данных профилирования:

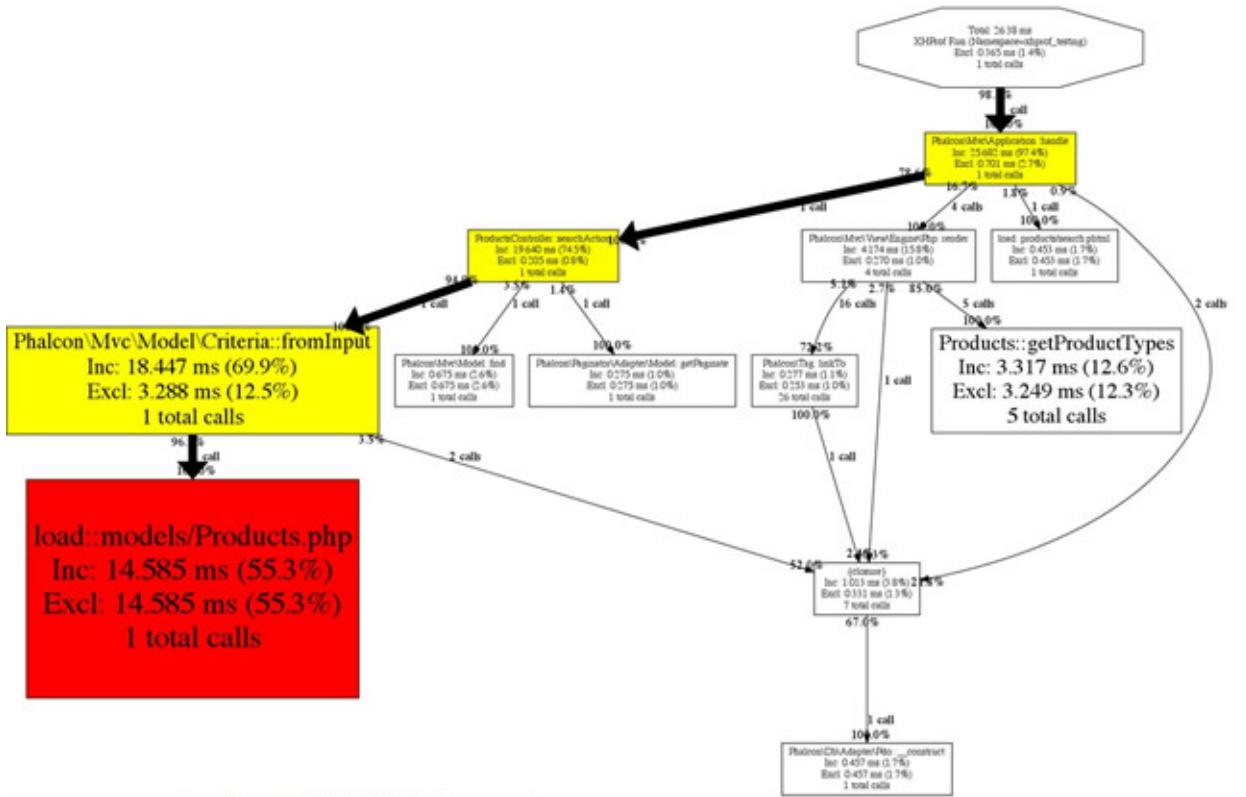
Displaying top 100 functions: Sorted by Excl. Wall Time (microsec) [[display all](#)]

| Function Name                                | Calls | Calls% | Incl. Wall Time (microsec) | IWall% | Excl. Wall Time (microsec) | EWall% | Incl. CPU (microsecs) | ICpu%  | Excl. CPU (microsec) | ECPU% | Incl. MemUse (bytes) |
|--|-------|--------|----------------------------|--------|----------------------------|--------|-----------------------|--------|----------------------|-------|----------------------|
| load:models/Products.php                     | 1     | 0.8%   | 14,585                     | 55.3%  | 14,585                     | 55.3%  | 346                   | 5.4%   | 346                  | 5.4%  | 43,528               |
| Phalcon\Mvc\Model\Criteria::fromInput        | 1     | 0.8%   | 18,447                     | 69.9%  | 3,288                      | 12.5%  | 1,515                 | 23.7%  | 831                  | 13.0% | 87,544               |
| Products::getProductTypes                    | 5     | 3.8%   | 3,317                      | 12.6%  | 3,249                      | 12.3%  | 1,029                 | 16.1%  | 958                  | 15.0% | 57,992               |
| Phalcon\Mvc\Application::handle              | 1     | 0.8%   | 25,682                     | 97.4%  | 701                        | 2.7%   | 5,981                 | 93.4%  | 683                  | 10.7% | 355,296              |
| Phalcon\Mvc\Model::find                      | 1     | 0.8%   | 675                        | 2.6%   | 675                        | 2.6%   | 457                   | 7.1%   | 457                  | 7.1%  | 16,288               |
| Phalcon\Db\Adapter\Pdo::__construct          | 1     | 0.8%   | 457                        | 1.7%   | 457                        | 1.7%   | 217                   | 3.4%   | 217                  | 3.4%  | 8,664                |
| load:products/search.php.html                | 1     | 0.8%   | 453                        | 1.7%   | 453                        | 1.7%   | 288                   | 4.5%   | 288                  | 4.5%  | 26,144               |
| main()                                       | 1     | 0.8%   | 26,380                     | 100.0% | 365                        | 1.4%   | 6,401                 | 100.0% | 128                  | 2.0%  | 402,568              |
| {closure}                                    | 7     | 5.4%   | 1,013                      | 3.8%   | 331                        | 1.3%   | 777                   | 12.1%  | 308                  | 4.8%  | 99,544               |
| Phalcon\Paginator\Adapter\Model::getPaginate | 1     | 0.8%   | 275                        | 1.0%   | 275                        | 1.0%   | 276                   | 4.3%   | 276                  | 4.3%  | 24,928               |
| Phalcon\Mvc\View\Engine\Php::render          | 4     | 3.1%   | 4,174                      | 15.8%  | 270                        | 1.0%   | 1,864                 | 29.1%  | 231                  | 3.6%  | 99,664               |
| Phalcon\Tag::linkTo                          | 26    | 20.0%  | 277                        | 1.1%   | 253                        | 1.0%   | 292                   | 4.6%   | 268                  | 4.2%  | 9,024                |
| ProductsController::searchAction             | 1     | 0.8%   | 19,640                     | 74.5%  | 205                        | 0.8%   | 2,479                 | 38.7%  | 184                  | 2.9%  | 142,392              |
| Phalcon\Loader::autoLoad                     | 5     | 3.8%   | 174                        | 0.7%   | 128                        | 0.5%   | 122                   | 1.9%   | 89                   | 1.4%  | 14,032               |
| Phalcon\Config\Adapter\Ini::__construct      | 1     | 0.8%   | 122                        | 0.5%   | 122                        | 0.5%   | 123                   | 1.9%   | 123                  | 1.9%  | 4,480                |
| Phalcon\Session\Adapter::start               | 1     | 0.8%   | 115                        | 0.4%   | 115                        | 0.4%   | 117                   | 1.8%   | 117                  | 1.8%  | 34,328               |
| Phalcon\DI::set                              | 8     | 6.2%   | 80                         | 0.3%   | 80                         | 0.3%   | 65                    | 1.0%   | 65                   | 1.0%  | 2,592                |
| Elements::getMenu                            | 1     | 0.8%   | 107                        | 0.4%   | 59                         | 0.2%   | 108                   | 1.7%   | 52                   | 0.8%  | 5,632                |
| Security::beforeDispatch                     | 1     | 0.8%   | 243                        | 0.9%   | 52                         | 0.2%   | 244                   | 3.8%   | 46                   | 0.7%  | 47,208               |
| Phalcon\Loader::__construct                  | 1     | 0.8%   | 51                         | 0.2%   | 51                         | 0.2%   | 15                    | 0.2%   | 15                   | 0.2%  | 1,160                |
| Elements::getTabs                            | 1     | 0.8%   | 96                         | 0.4%   | 51                         | 0.2%   | 97                    | 1.5%   | 45                   | 0.7%  | 4,504                |
| Phalcon\Di\FactoryDefault::__construct       | 1     | 0.8%   | 47                         | 0.2%   | 47                         | 0.2%   | 49                    | 0.8%   | 49                   | 0.8%  | 17,736               |

## Профилирование SQL-запросов

Большинство систем баз данных предоставляет средства для выявления медленных SQL запросов. Обнаружение и исправление медленных запросов очень важно для увеличения производительности на стороне сервера. В случае с MySQL вы можете использовать лог медленных запросов (slow query log), чтобы понять, какие SQL-запросы выполняются медленнее, чем ожидалось:

```
log-slow-queries = /var/log/slow-queries.log
long_query_time = 1.5
```



## 2.52.2 Профилирование на клиенте

Иногда, для увеличения производительности, может понадобиться проанализировать приложение и веб-сервер для улучшения загрузки статических элементов, таких как картинки, javascript и css. Следующие инструменты полезны для выявления общих узких мест на стороне клиента:

### Профилирование в Chrome/Firefox

У большинства современных браузеров есть все инструменты для профилирования загрузки страницы. В Chrome для получения информации о загрузке различных ресурсов можно использовать веб-инспектор:

Firebug обеспечивает схожую функциональность:

## 2.52.3 Использование Yahoo! YSlow

YSlow анализирует веб-страницу и показывает советы по улучшению производительности на основе комплекса правил для высокопроизводительных веб-страниц

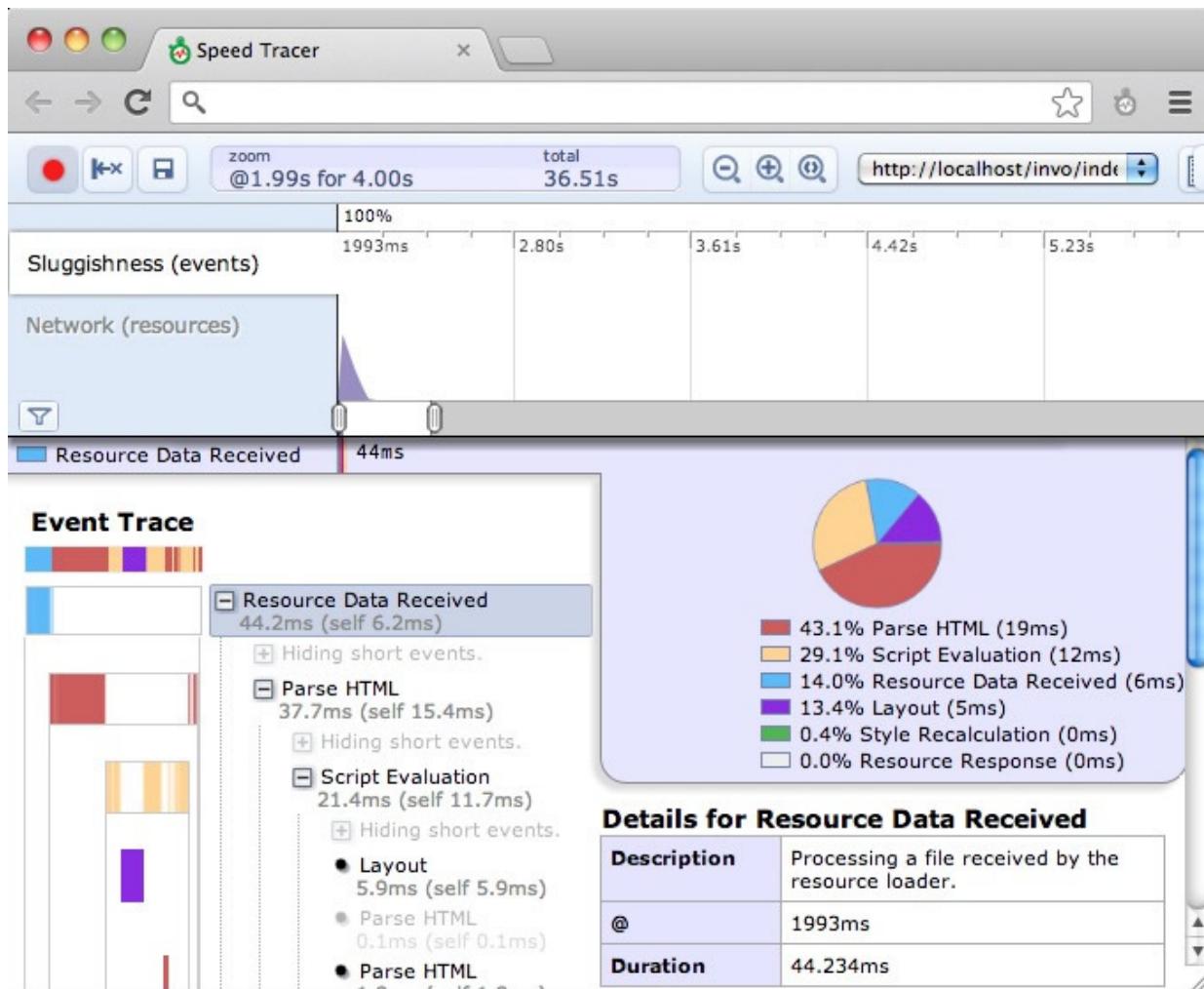
### Использование Speed Tracer

Speed Tracer - инструмент, помогающий обнаружить и устранить проблемы производительности в клиентской части веб-приложения. Он показывает метрики, полученные из работы браузера на самом низком уровне. Speed Racer доступен как расширение для Google Chrome и работает на всех поддерживаемых в настоящее время платформах (Windows и Linux).









Этот инструмент очень полезен, так как он позволяет получать в реальном времени параметры, используемые для отображения полной страницы, включая разбор HTML, оценку Javascript и CSS стилей.

#### 2.52.4 Использование акселераторов PHP

APC, как и многие другие PHP акселераторы, помогает приложению: уменьшает накладные расходы на чтение, разбивку и разбор PHP файлов при каждом запросе. После установки расширения используйте следующий параметр, чтобы включить APC:

```
apc.enabled = On
```

PHP 5.5 включает в себя встроенный кешер байт-кода под названием ZendOptimizer+, это расширение также доступно для версий 5.3 и 5.4.

#### 2.52.5 Модуль Google Page Speed

mod\_pagespeed увеличивает скорость вашего сайта и уменьшает время загрузки страницы. Это open-source модуль для HTTP-сервера Apache, он использует для повышения производительности лучшие практики обработки страниц и соответствующих ресурсов (CSS, JavaScript, изображений), не требуя ручной модификации уже существующего контента или настроек.

### 2.53 Модульное тестирование (Unit test)

Написание качественных тестов может помочь в создании качественного программного обеспечения. Если вы используете модульное тестирование, вы можете избежать большое количество ошибок и упростить поддержку программного обеспечения.

#### 2.53.1 Интеграция Phalcon с PHPUnit

Если вы еще не установили PHPUnit, вы можете сделать это с помощью следующей команды composer:

```
composer require phpunit/phpunit
```

или вручную добавить его в composer.json:

```
{
    "require-dev": {
        "phpunit/phpunit": "3.7.*"
    }
}
```

Или, если у вас нет composer, можно установить с помощью pear PHPUnit:

```
pear config-set auto_discover 1
pear install pear.phpunit.de/PHPUnit
```

После установки PHPUnit создайте директорию tests' в корне проекта:

```
app/
public/
tests/
```

Далее, нам понадобится файл “загрузчик” для подготовки приложения модульного тестирования.

## 2.53.2 Загрузчик PHPPunit

Файл загрузчик необходим для подготовки приложения к запуску тестов. Мы подготовили образец файла. Поместите файл TestHelper.php в /tests.

```
<?php
use Phalcon\DI,
    Phalcon\DI\FactoryDefault;

ini_set('display_errors',1);
error_reporting(E_ALL);

define('ROOT_PATH', __DIR__);
define('PATH_LIBRARY', __DIR__ . '/../app/library/');
define('PATH_SERVICES', __DIR__ . '/../app/services/');
define('PATH_RESOURCES', __DIR__ . '/../app/resources/');

set_include_path(
    ROOT_PATH . PATH_SEPARATOR . get_include_path()
);

// требуется для phalcon/incubator
include __DIR__ . "/../vendor/autoload.php";

// Используем автозагрузчик приложений для автозагрузки классов.
// Автозагрузка зависимостей, найденных в composer.
$loader = new \Phalcon\Loader();

$loader->registerDirs(array(
    ROOT_PATH
));

$loader->register();

$di = new FactoryDefault();
DI::reset();

// здесь можно добавить любые необходимые сервисы в контейнер зависимостей
DI::setDefault($di);
```

Если вам необходимо протестировать любой компонент из вашей библиотеки, добавьте их в автозагрузку или используйте загрузчик вашего основного приложения.

Чтобы помочь вам построить юнит-тесты, мы сделали несколько абстрактных классов, которые вы можете использовать для загрузки самих тестов. Вы можете взять их в репозитарии инкубатора Phalcon @ <https://github.com/phalcon/incubator>.

Вы можете использовать инкубатор, добавив его в зависимости composer:

```
composer require phalcon/incubator
```

или вручную добавить его в composer.json:

```
{
    "require": {
        "phalcon/incubator": "dev-master"
    }
}
```

Вы также можете клонировать репозиторий, используя ссылку выше.

### 2.53.3 Файл PHPUnit.xml

Теперь создайте phpunit файл:

```
<?xml version="1.0" encoding="UTF-8"?>
<phpunit bootstrap=". /TestHelper.php"
    backupGlobals="false"
    backupStaticAttributes="false"
    verbose="true"
    colors="false"
    convertErrorsToExceptions="true"
    convertNoticesToExceptions="true"
    convertWarningsToExceptions="true"
    processIsolation="false"
    stopOnFailure="false"
    syntaxCheck="true">
    <testsuite name="Phalcon - Testsuite">
        <directory>./</directory>
    </testsuite>
</phpunit>
```

Измените phpunit.xml в соответствии с вашими потребностями и сохраните его в tests/.

### 2.53.4 Пример юнит-теста

Для работы с юнит-тестом необходимо его определить. Автозагрузчик сам будет загружать все созданные вами файлы и передавать из PHPUnit для выполнения тестов. Таким образом, вам необходимо будет только создать файлы, а PHPUnit будет запускать тесты для вас.

Этот пример не содержит конфигурационного файла, хотя в большинстве случаев без него не обойтись в тестах. Вы можете добавить его в DI и получить его файле UnitTestCase.

Сначала создайте базовый файл для ваших юнит-тестов UnitTestCase.php в папке /tests:

```
<?php
use Phalcon\DI,
    \Phalcon\Test\UnitTestCase as PhalconTestCase;

abstract class UnitTestCase extends PhalconTestCase {

    /**
     * @var \Voice\Cache
     */
    protected $_cache;

    /**
     * @var \Phalcon\Config
     */
    protected $_config;

    /**
     * @var bool
     */
    private $_loaded = false;
```

```

public function setUp(Phalcon\DiInterface $di = NULL, Phalcon\Config $config = NULL) {

    // Загрузка дополнительных сервисов, которые могут потребоваться во время тестирования
    $di = DI::getDefault();

    // получаем любые компоненты DI, если у вас есть настройки, не забудьте передать их родителю
    parent::setUp($di);

    $this->_loaded = true;
}

/**
 * Проверка на то, что тест правильно настроен
 * @throws \PHPUnit_Framework_IncompleteTestError;
 */
public function __destruct() {
    if(!$this->_loaded) {
        throw new \PHPUnit_Framework_IncompleteTestError('Please run parent::setUp().');
    }
}
}

```

Хорошая идея: разделять юнит-тесты в пространствах имен. Для этого теста мы создадим пространство имен ‘Test’. Создайте файл с названием tests/TestUnit.php:

```

<?php
namespace Test;
/**
 * Class UnitTest
 */
class UnitTest extends \TestCase {

    public function testTestCase() {

        $this->assertEquals('works',
            'works',
            'This is OK'
        );

        $this->assertEquals('works',
            'works1',
            'This will fail'
        );
    }
}

```

После выполнения ‘phpunit’ в командной строке в каталоге tests вы получите следующий результат:

```

$ phpunit
PHPUnit 3.7.23 by Sebastian Bergmann.

Configuration read from /private/var/www/tests/phpunit.xml

Time: 3 ms, Memory: 3.25Mb

```

There was 1 failure:

```
1) Test\UnitTest::testTestCase
This wil fail
Failed asserting that two strings are equal.
--- Expected
+++ Actual
@@ @@
-'works'
+'works1'
```

/private/var/www/tests/Test/UnitTest.php:25

FAILURES!

Tests: 1, Assertions: 2, Failures: 1.

Теперь вы можете начать писать собственные юнит-тесты. Здесь находится хорошее руководство ( Мы рекомендуем вам ознакомиться с документацией PHPUnit, если вы ещё не знакомы с PHPUnit):

<http://blog.stevensanderson.com/2009/08/24/writing-great-unit-tests-best-and-worst-practises/>

## 2.54 API Indice

### 2.54.1 Abstract class Phalcon\Acl

This component allows to manage ACL lists. An access control list (ACL) is a list of permissions attached to an object. An ACL specifies which users or system processes are granted access to objects, as well as what operations are allowed on given objects.

```
<?php

$acl = new Phalcon\Acl\Adapter\Memory();

//Default action is deny access
$acl->setDefaultAction(Phalcon\Acl::DENY);

//Create some roles
$roleAdmins = new Phalcon\Acl\Role('Administrators', 'Super-User role');
$roleGuests = new Phalcon\Acl\Role('Guests');

//Add "Guests" role to acl
$acl->addRole($roleGuests);

//Add "Designers" role to acl
$acl->addRole('Designers');

//Define the "Customers" resource
$customersResource = new Phalcon\Acl\Resource('Customers', 'Customers management');

//Add "customers" resource with a couple of operations
$acl->addResource($customersResource, 'search');
$acl->addResource($customersResource, array('create', 'update'));

//Set access level for roles into resources
$acl->allow('Guests', 'Customers', 'search');
```

```

$acl->allow('Guests', 'Customers', 'create');
$acl->deny('Guests', 'Customers', 'update');

//Check whether role has access to the operations
$acl->isAllowed('Guests', 'Customers', 'edit'); //Returns 0
$acl->isAllowed('Guests', 'Customers', 'search'); //Returns 1
$acl->isAllowed('Guests', 'Customers', 'create'); //Returns 1

```

## Constants

*integer* **ALLOW**

*integer* **DENY**

### 2.54.2 Abstract class Phalcon\Acl\Adapter

*implements Phalcon\Events\EventsAwareInterface, Phalcon\Acl\AdapterInterface*

Phalcon\Acl\Adapter initializer

## Methods

**public setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager)

Sets the events manager

**public Phalcon\Events\ManagerInterface getEventsManager** ()

Returns the internal event manager

**public setDefaultAction** (*int* \$defaultAccess)

Sets the default access level (Phalcon\Acl::ALLOW or Phalcon\Acl::DENY)

**public int getDefaultAction** ()

Returns the default ACL access level

**public string getActiveRole** ()

Returns the role which the list is checking if it's allowed to certain resource/access

**public string getActiveResource** ()

Returns the resource which the list is checking if some role can access it

**public string getActiveAccess** ()

Returns the access which the list is checking if some role can access it

**abstract public boolean addRole** (*Phalcon\Acl\RoleInterface* \$role, [*string* \$accessInherits]) inherited from Phalcon\Acl\AdapterInterface

Adds a role to the ACL list. Second parameter lets to inherit access data from other existing role

**abstract public addInherit** (*string* \$roleName, *string* \$roleToInherit) inherited from Phalcon\Acl\AdapterInterface

Do a role inherit from another existing role

**abstract public boolean isRole** (*string* \$roleName) inherited from Phalcon\Acl\AdapterInterface

Check whether role exist in the roles list

abstract public **boolean isResource** (*string \$resourceName*) inherited from Phalcon\Acl\AdapterInterface

Check whether resource exist in the resources list

abstract public **boolean addResource** (*Phalcon\Acl\ResourceInterface \$resource, [array \$accessList]*)  
inherited from Phalcon\Acl\AdapterInterface

Adds a resource to the ACL list Access names can be a particular action, by example search, update, delete, etc or a list of them

abstract public **addResourceAccess** (*string \$resourceName, mixed \$accessList*) inherited from Phalcon\Acl\AdapterInterface

Adds access to resources

abstract public **dropResourceAccess** (*string \$resourceName, mixed \$accessList*) inherited from Phalcon\Acl\AdapterInterface

Removes an access from a resource

abstract public **allow** (*string \$roleName, string \$resourceName, mixed \$access*) inherited from Phalcon\Acl\AdapterInterface

Allow access to a role on a resource

abstract public **deny** (*string \$roleName, string \$resourceName, mixed \$access*) inherited from Phalcon\Acl\AdapterInterface

Deny access to a role on a resource

abstract public **boolean isAllowed** (*string \$role, string \$resource, string \$access*) inherited from Phalcon\Acl\AdapterInterface

Check whether a role is allowed to access an action from a resource

abstract public *Phalcon\Acl\RoleInterface [] getRoles ()* inherited from Phalcon\Acl\AdapterInterface

Return an array with every role registered in the list

abstract public *Phalcon\Acl\ResourceInterface [] getResources ()* inherited from Phalcon\Acl\AdapterInterface

Return an array with every resource registered in the list

### 2.54.3 Class Phalcon\Acl\Adapter\Memory

*extends abstract class Phalcon\Acl\Adapter*

*implements Phalcon\Acl\AdapterInterface, Phalcon\Events\EventsAwareInterface*

Manages ACL lists in memory

<?php

```
$acl = new Phalcon\Acl\Adapter\Memory();

$acl->setDefaultAction(Phalcon\Acl::DENY);

//Register roles
$roles = array(
    'users' => new Phalcon\Acl\Role('Users'),
    'guests' => new Phalcon\Acl\Role('Guests')
```

```

);

foreach ($roles as $role) {
    $acl->addRole($role);
}

//Private area resources
$privateResources = array(
    'companies' => array('index', 'search', 'new', 'edit', 'save', 'create', 'delete'),
    'products' => array('index', 'search', 'new', 'edit', 'save', 'create', 'delete'),
    'invoices' => array('index', 'profile')
);
foreach ($privateResources as $resource => $actions) {
    $acl->addResource(new Phalcon\Acl\Resource($resource), $actions);
}

//Public area resources
$publicResources = array(
    'index' => array('index'),
    'about' => array('index'),
    'session' => array('index', 'register', 'start', 'end'),
    'contact' => array('index', 'send')
);
foreach ($publicResources as $resource => $actions) {
    $acl->addResource(new Phalcon\Acl\Resource($resource), $actions);
}

//Grant access to public areas to both users and guests
foreach ($roles as $role){
    foreach ($publicResources as $resource => $actions) {
        $acl->allow($role->getName(), $resource, '*');
    }
}

//Grant access to private area to role Users
foreach ($privateResources as $resource => $actions) {
    foreach ($actions as $action) {
        $acl->allow('Users', $resource, $action);
    }
}

```

## Methods

**public \_\_construct ()**

Phalcon\Acl\Adapter\Memory constructor

**public boolean addRole (*Phalcon\Acl\RoleInterface* \$role, [*array/string* \$accessInherits])**

Adds a role to the ACL list. Second parameter allows inheriting access data from other existing role Example:

<?php

```

$acl->addRole(new Phalcon\Acl\Role('administrator'), 'consultant');
$acl->addRole('administrator', 'consultant');

```

**public addInherit (*string* \$roleName, *string* \$roleToInherit)**

Do a role inherit from another existing role

public *boolean* **isRole** (*string* \$roleName)

Check whether role exist in the roles list

public *boolean* **isResource** (*string* \$resourceName)

Check whether resource exist in the resources list

public *boolean* **addResource** (*Phalcon\Acl\Resource* \$resource, [*array* \$accessList])

Adds a resource to the ACL list Access names can be a particular action, by example search, update, delete, etc or a list of them Example:

<?php

```
//Add a resource to the the list allowing access to an action
$acl->addResource(new Phalcon\Acl\Resource('customers'), 'search');
$acl->addResource('customers', 'search');
```

```
//Add a resource with an access list
$acl->addResource(new Phalcon\Acl\Resource('customers'), array('create', 'search'));
$acl->addResource('customers', array('create', 'search'));
```

public **addResourceAccess** (*string* \$resourceName, *mixed* \$accessList)

Adds access to resources

public **dropResourceAccess** (*string* \$resourceName, *mixed* \$accessList)

Removes an access from a resource

protected **\_allowOrDeny** ()

Checks if a role has access to a resource

public **allow** (*string* \$roleName, *string* \$resourceName, *mixed* \$access)

Allow access to a role on a resource You can use '\*' as wildcard Example:

<?php

```
//Allow access to guests to search on customers
$acl->allow('guests', 'customers', 'search');

//Allow access to guests to search or create on customers
$acl->allow('guests', 'customers', array('search', 'create'));

//Allow access to any role to browse on products
$acl->allow('*', 'products', 'browse');

//Allow access to any role to browse on any resource
$acl->allow('*', '*', 'browse');
```

public *boolean* **deny** (*string* \$roleName, *string* \$resourceName, *mixed* \$access)

Deny access to a role on a resource You can use '\*' as wildcard Example:

<?php

```
//Deny access to guests to search on customers
$acl->deny('guests', 'customers', 'search');

//Deny access to guests to search or create on customers
$acl->deny('guests', 'customers', array('search', 'create'));
```

```
//Deny access to any role to browse on products
$acl->deny('*', 'products', 'browse');

//Deny access to any role to browse on any resource
$acl->deny('*', '*', 'browse');

public boolean isAllowed (string $role, string $resource, string $access)
Check whether a role is allowed to access an action from a resource

<?php

//Does andres have access to the customers resource to create?
$acl->isAllowed('andres', 'Products', 'create');

//Do guests have access to any resource to edit?
$acl->isAllowed('guests', '*', 'edit');

public Phalcon\Acl\Role [] getRoles ()
Return an array with every role registered in the list

public Phalcon\Acl\Resource [] getResources ()
Return an array with every resource registered in the list

public setEventsManager (Phalcon\Events\ManagerInterface $eventsManager) inherited from
Phalcon\Acl\Adapter

Sets the events manager

public Phalcon\Events\ManagerInterface getEventsManager () inherited from Phalcon\Acl\Adapter

Returns the internal event manager

public setDefaultAction (int $defaultAccess) inherited from Phalcon\Acl\Adapter

Sets the default access level (Phalcon\Acl::ALLOW or Phalcon\Acl::DENY)

public int getDefaultAction () inherited from Phalcon\Acl\Adapter

Returns the default ACL access level

public string getActiveRole () inherited from Phalcon\Acl\Adapter

Returns the role which the list is checking if it's allowed to certain resource/access

public string getActiveResource () inherited from Phalcon\Acl\Adapter

Returns the resource which the list is checking if some role can access it

public string getActiveAccess () inherited from Phalcon\Acl\Adapter

Returns the access which the list is checking if some role can access it
```

#### 2.54.4 Class Phalcon\Acl\Exception

*extends class Phalcon\Exception*

Class for exceptions thrown by Phalcon\Acl

## Methods

final private *Exception* **\_\_clone** () inherited from *Exception*  
Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from *Exception*  
*Exception* constructor

final public *string* **getMessage** () inherited from *Exception*  
Gets the *Exception* message

final public *int* **getCode** () inherited from *Exception*  
Gets the *Exception* code

final public *string* **getFile** () inherited from *Exception*  
Gets the file in which the exception occurred

final public *int* **getLine** () inherited from *Exception*  
Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from *Exception*  
Gets the stack trace

final public *Exception* **getPrevious** () inherited from *Exception*  
Returns previous *Exception*

final public *Exception* **getTraceAsString** () inherited from *Exception*  
Gets the stack trace as a string

public *string* **\_\_toString** () inherited from *Exception*  
String representation of the exception

### 2.54.5 Class Phalcon\Acl\Resource

*implements Phalcon\Acl\ResourceInterface*

This class defines resource entity and its description

## Methods

public **\_\_construct** (*string* \$name, [*string* \$description])  
Phalcon\Acl\Resource constructor

public *string* **getName** ()  
Returns the resource name

public *string* **getDescription** ()  
Returns resource description

public *string* **\_\_toString** ()  
Magic method **\_\_toString**

## 2.54.6 Class Phalcon\Acl\Role

*implements Phalcon\Acl\RoleInterface*

This class defines role entity and its description

### Methods

public **\_\_construct** (*string \$name, [string \$description]*)

Phalcon\Acl\Role description

public *string getName ()*

Returns the role name

public *string getDescription ()*

Returns role description

public *string \_\_toString ()*

Magic method \_\_toString

## 2.54.7 Abstract class Phalcon\Annotations\Adapter

*implements Phalcon\Annotations\AdapterInterface*

This is the base class for Phalcon\Annotations adapters

### Methods

public **setReader** (*Phalcon\Annotations\ReaderInterface \$reader*)

Sets the annotations parser

public *Phalcon\Annotations\ReaderInterface getReader ()*

Returns the annotation reader

public *Phalcon\Annotations\Reflection get (string/object \$className)*

Parses or retrieves all the annotations found in a class

public *array getMethods (string \$className)*

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection getMethod (string \$className, string \$methodName)*

Returns the annotations found in a specific method

public *array getProperties (string \$className)*

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection getProperty (string \$className, string \$propertyName)*

Returns the annotations found in a specific property

abstract public *Phalcon\Annotations\Reflection read (string \$key)* inherited from  
Phalcon\Annotations\AdapterInterface

Read parsed annotations

abstract public **write** (string \$key, *Phalcon\Annotations\Reflection* \$data) inherited from Phalcon\Annotations\AdapterInterface

Write parsed annotations

## 2.54.8 Class Phalcon\Annotations\Adapter\Apc

*extends abstract class Phalcon\Annotations\Adapter*

*implements Phalcon\Annotations\AdapterInterface*

Stores the parsed annotations in APC. This adapter is suitable for production

<?php

```
$annotations = new \Phalcon\Annotations\Adapter\Apc();
```

### Methods

public *Phalcon\Annotations\Reflection* **read** (string \$key)

Reads parsed annotations from APC

public **write** (string \$key, *Phalcon\Annotations\Reflection* \$data)

Writes parsed annotations to APC

public **setReader** (*Phalcon\Annotations\ReaderInterface* \$reader) inherited from Phalcon\Annotations\Adapter

Sets the annotations parser

public *Phalcon\Annotations\ReaderInterface* **getReader** () inherited from Phalcon\Annotations\Adapter

Returns the annotation reader

public *Phalcon\Annotations\Reflection* **get** (string/object \$className) inherited from Phalcon\Annotations\Adapter

Parses or retrieves all the annotations found in a class

public array **getMethods** (string \$className) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getMethod** (string \$className, string \$methodName) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in a specific method

public array **getProperties** (string \$className) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getProperty** (string \$className, string \$propertyName) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in a specific property

## 2.54.9 Class Phalcon\Annotations\Adapter\Files

*extends abstract class Phalcon\Annotations\Adapter*

*implements Phalcon\Annotations\AdapterInterface*

Stores the parsed annotations in files. This adapter is suitable for production

<?php

```
$annotations = new \Phalcon\Annotations\Adapter\Files(array(
    'annotationsDir' => 'app/cache/annotations/',
));
```

### Methods

public **\_\_construct** ([array \$options])

Phalcon\Annotations\Adapter\Files constructor

public *Phalcon\Annotations\Reflection* **read** (string \$key)

Reads parsed annotations from files

public **write** (string \$key, *Phalcon\Annotations\Reflection* \$data)

Writes parsed annotations to files

public **setReader** (*Phalcon\Annotations\ReaderInterface* \$reader) inherited from Phalcon\Annotations\Adapter

Sets the annotations parser

public *Phalcon\Annotations\ReaderInterface* **getReader** () inherited from Phalcon\Annotations\Adapter

Returns the annotation reader

public *Phalcon\Annotations\Reflection* **get** (string/object \$className) inherited from Phalcon\Annotations\Adapter

Parses or retrieves all the annotations found in a class

public array **getMethods** (string \$className) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getMethod** (string \$className, string \$methodName) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in a specific method

public array **getProperties** (string \$className) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getProperty** (string \$className, string \$propertyName) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in a specific property

## 2.54.10 Class Phalcon\Annotations\Adapter\Memory

*extends abstract class Phalcon\Annotations\Adapter*

*implements Phalcon\Annotations\AdapterInterface*

Stores the parsed annotations in memory. This adapter is the suitable development/testing

### Methods

public *Phalcon\Annotations\Reflection* **read** (*string \$key*)

Reads parsed annotations from memory

public **write** (*string \$key, Phalcon\Annotations\Reflection \$data*)

Writes parsed annotations to memory

public **setReader** (*Phalcon\Annotations\ReaderInterface \$reader*) inherited from Phalcon\Annotations\Adapter

Sets the annotations parser

public *Phalcon\Annotations\ReaderInterface* **getReader** () inherited from Phalcon\Annotations\Adapter

Returns the annotation reader

public *Phalcon\Annotations\Reflection* **get** (*string/object \$className*) inherited from Phalcon\Annotations\Adapter

Parses or retrieves all the annotations found in a class

public *array* **getMethods** (*string \$className*) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getMethod** (*string \$className, string \$methodName*) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in a specific method

public *array* **getProperties** (*string \$className*) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getProperty** (*string \$className, string \$propertyName*) inherited from Phalcon\Annotations\Adapter

Returns the annotations found in a specific property

## 2.54.11 Class Phalcon\Annotations\Adapter\Xcache

*extends abstract class Phalcon\Annotations\Adapter*

*implements Phalcon\Annotations\AdapterInterface*

Stores the parsed annotations to XCache. This adapter is suitable for production

<?php

```
$annotations = new \Phalcon\Annotations\Adapter\Xcache();
```

## Methods

public *Phalcon\Annotations\Reflection* **read** (*string* \$key)

Reads parsed annotations from XCache

public **write** (*string* \$key, *Phalcon\Annotations\Reflection* \$data)

Writes parsed annotations to XCache

public **setReader** (*Phalcon\Annotations\ReaderInterface* \$reader) inherited from *Phalcon\Annotations\Adapter*

Sets the annotations parser

public *Phalcon\Annotations\ReaderInterface* **getReader** () inherited from *Phalcon\Annotations\Adapter*

Returns the annotation reader

public *Phalcon\Annotations\Reflection* **get** (*string/object* \$className) inherited from *Phalcon\Annotations\Adapter*

Parses or retrieves all the annotations found in a class

public *array* **getMethods** (*string* \$className) inherited from *Phalcon\Annotations\Adapter*

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getMethod** (*string* \$className, *string* \$methodName) inherited from *Phalcon\Annotations\Adapter*

Returns the annotations found in a specific method

public *array* **getProperties** (*string* \$className) inherited from *Phalcon\Annotations\Adapter*

Returns the annotations found in all the class' methods

public *Phalcon\Annotations\Collection* **getProperty** (*string* \$className, *string* \$propertyName) inherited from *Phalcon\Annotations\Adapter*

Returns the annotations found in a specific property

## 2.54.12 Class Phalcon\Annotations\Annotation

Represents a single annotation in an annotations collection

## Methods

public **\_\_construct** (*array* \$reflectionData)

*Phalcon\Annotations\Annotation* constructor

public *string* **getName** ()

Returns the annotation's name

public *mixed* **getExpression** (*array* \$expr)

Resolves an annotation expression

public *array* **getExprArguments** ()

Returns the expression arguments without resolving

public *array* **getArguments** ()

Returns the expression arguments

`public int numberArguments ()`

Returns the number of arguments that the annotation has

`public mixed getArgument (unknown $position)`

Returns an argument in a specific position

`public bool hasArgument (unknown $position)`

Checks if the annotation has a specific argument

`public mixed getNamedArgument (unknown $position)`

Returns a named argument

`public mixed getNamedParameter (unknown $position)`

Returns a named argument (deprecated)

`public boolean hasNamedArgument (unknown $position)`

Checks if the annotation has a specific named argument

### 2.54.13 Class Phalcon\Annotations\Collection

*implements* Iterator, Traversable, Countable

Represents a collection of annotations. This class allows to traverse a group of annotations easily

`<?php`

```
//Traverse annotations
foreach ($classAnnotations as $annotation) {
    echo 'Name=', $annotation->getName(), PHP_EOL;
}

//Check if the annotations has a specific
var_dump($classAnnotations->has('Cacheable'));

//Get an specific annotation in the collection
$annotation = $classAnnotations->get('Cacheable');
```

#### Methods

`public __construct ([array $reflectionData])`

Phalcon\Annotations\Collection constructor

`public int count ()`

Returns the number of annotations in the collection

`public rewind ()`

Rewinds the internal iterator

`public Phalcon\Annotations\Annotation current ()`

Returns the current annotation in the iterator

`public int key ()`

Returns the current position/key in the iterator

**public `next ()`**

Moves the internal iteration pointer to the next position

**public `boolean valid ()`**

Check if the current annotation in the iterator is valid

**public `Phalcon\Annotations\Annotation [] getAnnotations ()`**

Returns the internal annotations as an array

**public `Phalcon\Annotations\Annotation get (string $name)`**

Returns the first annotation that match a name

**public `Phalcon\Annotations\Annotation [] getAll (string $name)`**

Returns all the annotations that match a name

**public `boolean has (string $name)`**

Check if an annotation exists in a collection

## 2.54.14 Class Phalcon\Annotations\Exception

*extends class `Phalcon\Exception`*

Exceptions thrown in Phalcon\Annotations will use this class

### Methods

**final private `Exception __clone ()`** inherited from Exception

Clone the exception

**public `__construct ([string $message], [int $code], [Exception $previous])`** inherited from Exception

Exception constructor

**final public `string getMessage ()`** inherited from Exception

Gets the Exception message

**final public `int getCode ()`** inherited from Exception

Gets the Exception code

**final public `string getFile ()`** inherited from Exception

Gets the file in which the exception occurred

**final public `int getLine ()`** inherited from Exception

Gets the line in which the exception occurred

**final public `array getTrace ()`** inherited from Exception

Gets the stack trace

**final public `Exception getPrevious ()`** inherited from Exception

Returns previous Exception

**final public `Exception getTraceAsString ()`** inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

## 2.54.15 Class Phalcon\Annotations\Reader

*implements Phalcon\Annotations\ReaderInterface*

Parses docblocks returning an array with the found annotations

### Methods

public *array* **parse** (*string* \$className)

Reads annotations from the class dockblocks, its methods and/or properties

public static *array* **parseDocBlock** (*string* \$docBlock, [*string* \$file], [*int* \$line])

Parses a raw doc block returning the annotations found

## 2.54.16 Class Phalcon\Annotations\Reflection

Allows to manipulate the annotations reflection in an OO manner

<?php

```
//Parse the annotations in a class
$reader = new \Phalcon\Annotations\Reader();
$parsing = $reader->parse('MyComponent');

//Create the reflection
$reflection = new \Phalcon\Annotations\Reflection($parsing);

//Get the annotations in the class docblock
$classAnnotations = $reflection->getClassAnnotations();
```

### Methods

public **\_\_construct** ([*array* \$reflectionData])

Phalcon\Annotations\Reflection constructor

public *Phalcon\Annotations\Collection* **getClassAnnotations** ()

Returns the annotations found in the class docblock

public *Phalcon\Annotations\Collection* [] **getMethodsAnnotations** ()

Returns the annotations found in the methods' docblocks

public *Phalcon\Annotations\Collection* [] **getPropertiesAnnotations** ()

Returns the annotations found in the properties' docblocks

public *array* **getReflectionData** ()

Returns the raw parsing intermediate definitions used to construct the reflection

```
public static array __set_state (unknown $data)
Restores the state of a Phalcon\Annotations\Reflection variable export
```

## 2.54.17 Class Phalcon\Assets\Collection

*implements* Countable, Iterator, Traversable

Represents a collection of resources

### Methods

```
public Phalcon\Assets\Collection add (Phalcon\Assets\Resource $resource)
```

Adds a resource to the collection

```
public Phalcon\Assets\Collection addCss (string $path, [boolean $local], [boolean $filter], [array $attributes])
```

Adds a CSS resource to the collection

```
public Phalcon\Assets\Collection addJs (string $path, [boolean $local], [boolean $filter], [array $attributes])
```

Adds a javascript resource to the collection

```
public Phalcon\Assets\Resource [] getResources ()
```

Returns the resources as an array

```
public int count ()
```

Returns the number of elements in the form

```
public rewind ()
```

Rewinds the internal iterator

```
public Phalcon\Assets\Resource current ()
```

Returns the current resource in the iterator

```
public int key ()
```

Returns the current position/key in the iterator

```
public next ()
```

Moves the internal iteration pointer to the next position

```
public boolean valid ()
```

Check if the current element in the iterator is valid

```
public Phalcon\Assets\Collection setTargetPath (string $targetPath)
```

Sets the target path of the file for the filtered/join output

```
public string getTargetPath ()
```

Returns the target path of the file for the filtered/join output

```
public Phalcon\Assets\Collection setSourcePath (string $sourcePath)
```

Sets a base source path for all the resources in this collection

```
public string getSourcePath ()
```

Returns the base source path for all the resources in this collection

`public Phalcon\Assets\Collection setTargetUri (string $targetUri)`

Sets a target uri for the generated HTML

`public string getTargetUri ()`

Returns the target uri for the generated HTML

`public Phalcon\Assets\Collection setPrefix (string $prefix)`

Sets a common prefix for all the resources

`public string getPrefix ()`

Returns the prefix

`public Phalcon\Assets\Collection setLocal (boolean $local)`

Sets if the collection uses local resources by default

`public boolean getLocal ()`

Returns if the collection uses local resources by default

`public $this setAttributes (array $attributes)`

Sets extra HTML attributes

`public array getAttributes ()`

Returns extra HTML attributes

`public Phalcon\Assets\Collection addFilter (Phalcon\Assets\FilterInterface $filter)`

Adds a filter to the collection

`public Phalcon\Assets\Collection setFilters (array $filters)`

Sets an array of filters in the collection

`public array getFilters ()`

Returns the filters set in the collection

`public Phalcon\Assets\Collection join (boolean $join)`

Sets if all filtered resources in the collection must be joined in a single result file

`public boolean getJoin ()`

Returns if all the filtered resources must be joined

`public string getRealTargetPath ([string $basePath])`

Returns the complete location where the joined/filtered collection must be written

`public Phalcon\Assets\Collection setTargetLocal (boolean $targetLocal)`

Sets the target local

`public boolean getTargetLocal ()`

Returns the target local

## 2.54.18 Class Phalcon\Assets\Exception

*extends class Phalcon\Exception*

Phalcon\DI\Exception Exceptions thrown in Phalcon\Assets will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.19 Class Phalcon\Assets\Filters\Cssmin

*implements Phalcon\Assets\FilterInterface*

Minify the css - removes comments removes newlines and line feeds keeping removes last semicolon from last property

### Methods

public *\$content filter (string \$content)*

Filters the content using CSSMIN

## 2.54.20 Class Phalcon\Assets\Filters\Jsmin

*implements Phalcon\Assets\FilterInterface*

Deletes the characters which are insignificant to JavaScript. Comments will be removed. Tabs will be replaced with spaces. Carriage returns will be replaced with linefeeds. Most spaces and linefeeds will be removed.

### Methods

public **\$content filter (string \$content)**

Filters the content using JSMIN

## 2.54.21 Class Phalcon\Assets\Filters\None

*implements Phalcon\Assets\FilterInterface*

Returns the content without make any modification to the original source

### Methods

public **\$content filter (string \$content)**

Returns the content without touching

## 2.54.22 Class Phalcon\Assets\Manager

Manages collections of CSS/Javascript assets

### Methods

public **\_\_construct ([array \$options])**

Phalcon\Assets\Manager constructor

public *Phalcon\Assets\Manager* **setOptions (array \$options)**

Sets the manager's options

public **array getOptions ()**

Returns the manager's options

public *Phalcon\Assets\Manager* **useImplicitOutput (boolean \$implicitOutput)**

Sets if the HTML generated must be directly printed or returned

public *Phalcon\Assets\Manager* **addCss (string \$path, [boolean \$local], [boolean \$filter], [array \$attributes])**

Adds a Css resource to the 'css' collection

<?php

```
$assets->addCss('css/bootstrap.css');
$assets->addCss('http://bootstrap.my-cdn.com/style.css', false);
```

public *Phalcon\Assets\Manager* **addJs** (*string* \$path, [*boolean* \$local], [*boolean* \$filter], [*array* \$attributes])  
 Adds a javascript resource to the ‘js’ collection

<?php

```
$assets->addJs('scripts/jquery.js');
$assets->addJs('http://jquery.my-cdn.com/jquery.js', true);
```

public *Phalcon\Assets\Manager* **addResourceByType** (*string* \$type, *Phalcon\Assets\Resource* \$resource)  
 Adds a resource by its type

<?php

```
$assets->addResourceByType('css', new Phalcon\Assets\Resource\Css('css/style.css'));
```

public *Phalcon\Assets\Manager* **addResource** (*Phalcon\Assets\Resource* \$resource)

Adds a raw resource to the manager

<?php

```
$assets->addResource(new Phalcon\Assets\Resource('css', 'css/style.css'));
```

public *Phalcon\Assets\Manager* **set** (*string* \$id, *Phalcon\Assets\Collection* \$collection)

Sets a collection in the Assets Manager

<?php

```
$assets->get('js', $collection);
```

public *Phalcon\Assets\Collection* **get** (*string* \$id)

Returns a collection by its id

<?php

```
$scripts = $assets->get('js');
```

public *Phalcon\Assets\Collection* **getCss** ()

Returns the CSS collection of assets

public *Phalcon\Assets\Collection* **getJs** ()

Returns the CSS collection of assets

public *Phalcon\Assets\Collection* **collection** (*string* \$name)

Creates/Returns a collection of resources

public **output** (*Phalcon\Assets\Collection* \$collection, *callback* \$callback, [*string* \$type])

Traverses a collection calling the callback to generate its HTML

public **outputCss** ([*string* \$collectionName])

Prints the HTML for CSS resources

public **outputJs** ([*string* \$collectionName])

Prints the HTML for JS resources

## 2.54.23 Class Phalcon\Assets\Resource

Represents an asset resource

<?php

```
$resource = new Phalcon\Assets\Resource('js', 'javascripts/jquery.js');
```

### Methods

public **\_\_construct** (*string* \$type, *string* \$path, [*boolean* \$local], [*boolean* \$filter], [*array* \$attributes])

Phalcon\Assets\Resource constructor

public *Phalcon\Assets\Resource* **setType** (*string* \$type)

Sets the resource's type

public *string* **getType** ()

Returns the type of resource

public *Phalcon\Assets\Resource* **setPath** (*string* \$path)

Sets the resource's path

public *string* **getPath** ()

Returns the URI/URL path to the resource

public *Phalcon\Assets\Resource* **setLocal** (*boolean* \$local)

Sets if the resource is local or external

public *boolean* **getLocal** ()

Returns whether the resource is local or external

public *Phalcon\Assets\Resource* **setFilter** (*boolean* \$filter)

Sets if the resource must be filtered or not

public *boolean* **getFilter** ()

Returns whether the resource must be filtered or not

public *Phalcon\Assets\Resource* **setAttributes** (*array* \$attributes)

Sets extra HTML attributes

public *array* **getAttributes** ()

Returns extra HTML attributes set in the resource

public *Phalcon\Assets\Resource* **setTargetUri** (*string* \$targetUri)

Sets a target uri for the generated HTML

public *string* **getTargetUri** ()

Returns the target uri for the generated HTML

public *Phalcon\Assets\Resource* **setSourcePath** (*string* \$sourcePath)

Sets the resource's source path

public *string* **getSourcePath** ()

Returns the resource's target path

`public Phalcon\Assets\Resource setTargetPath (string $targetPath)`

Sets the resource's target path

`public string getTargetPath ()`

Returns the resource's target path

`public string getContent ([string $basePath])`

Returns the content of the resource as an string Optionally a base path where the resource is located can be set

`public string getRealTargetUri ()`

Returns the real target uri for the generated HTML

`public string getRealSourcePath ([string $basePath])`

Returns the complete location where the resource is located

`public string getRealTargetPath ([string $basePath])`

Returns the complete location where the resource must be written

## 2.54.24 Class Phalcon\Assets\Resource\Css

*extends class Phalcon\Assets\Resource*

Represents CSS resources

### Methods

`public __construct (string $path, [boolean $local], [boolean $filter], [array $attributes])`

`public Phalcon\Assets\Resource setType (string $type)` inherited from Phalcon\Assets\Resource

Sets the resource's type

`public string getType ()` inherited from Phalcon\Assets\Resource

Returns the type of resource

`public Phalcon\Assets\Resource setPath (string $path)` inherited from Phalcon\Assets\Resource

Sets the resource's path

`public string getPath ()` inherited from Phalcon\Assets\Resource

Returns the URI/URL path to the resource

`public Phalcon\Assets\Resource setLocal (boolean $local)` inherited from Phalcon\Assets\Resource

Sets if the resource is local or external

`public boolean getLocal ()` inherited from Phalcon\Assets\Resource

Returns whether the resource is local or external

`public Phalcon\Assets\Resource setFilter (boolean $filter)` inherited from Phalcon\Assets\Resource

Sets if the resource must be filtered or not

`public boolean getFilter ()` inherited from Phalcon\Assets\Resource

Returns whether the resource must be filtered or not

```
public Phalcon\Assets\Resource setAttributes (array $attributes) inherited from  
Phalcon\Assets\Resource
```

Sets extra HTML attributes

```
public array getAttributes () inherited from Phalcon\Assets\Resource
```

Returns extra HTML attributes set in the resource

```
public Phalcon\Assets\Resource setTargetUri (string $targetUri) inherited from Phalcon\Assets\Resource
```

Sets a target uri for the generated HTML

```
public string getTargetUri () inherited from Phalcon\Assets\Resource
```

Returns the target uri for the generated HTML

```
public Phalcon\Assets\Resource setSourcePath (string $sourcePath) inherited from  
Phalcon\Assets\Resource
```

Sets the resource's source path

```
public string getSourcePath () inherited from Phalcon\Assets\Resource
```

Returns the resource's target path

```
public Phalcon\Assets\Resource setTargetPath (string $targetPath) inherited from  
Phalcon\Assets\Resource
```

Sets the resource's target path

```
public string getTargetPath () inherited from Phalcon\Assets\Resource
```

Returns the resource's target path

```
public string getContent ([string $basePath]) inherited from Phalcon\Assets\Resource
```

Returns the content of the resource as an string Optionally a base path where the resource is located can be set

```
public string getRealTargetUri () inherited from Phalcon\Assets\Resource
```

Returns the real target uri for the generated HTML

```
public string getRealSourcePath ([string $basePath]) inherited from Phalcon\Assets\Resource
```

Returns the complete location where the resource is located

```
public string getRealTargetPath ([string $basePath]) inherited from Phalcon\Assets\Resource
```

Returns the complete location where the resource must be written

## 2.54.25 Class Phalcon\Assets\Resource\Js

*extends class Phalcon\Assets\Resource*

Represents Javascript resources

## Methods

public **\_\_construct** (*string \$path, [boolean \$local], [boolean \$filter], [array \$attributes]*)

public *Phalcon\Assets\Resource* **setType** (*string \$type*) inherited from *Phalcon\Assets\Resource*

Sets the resource's type

public *string getType* () inherited from *Phalcon\Assets\Resource*

Returns the type of resource

public *Phalcon\Assets\Resource* **setPath** (*string \$path*) inherited from *Phalcon\Assets\Resource*

Sets the resource's path

public *string getPath* () inherited from *Phalcon\Assets\Resource*

Returns the URI/URL path to the resource

public *Phalcon\Assets\Resource* **setLocal** (*boolean \$local*) inherited from *Phalcon\Assets\Resource*

Sets if the resource is local or external

public *boolean getLocal* () inherited from *Phalcon\Assets\Resource*

Returns whether the resource is local or external

public *Phalcon\Assets\Resource* **setFilter** (*boolean \$filter*) inherited from *Phalcon\Assets\Resource*

Sets if the resource must be filtered or not

public *boolean getFilter* () inherited from *Phalcon\Assets\Resource*

Returns whether the resource must be filtered or not

public *Phalcon\Assets\Resource* **setAttributes** (*array \$attributes*) inherited from *Phalcon\Assets\Resource*

Sets extra HTML attributes

public *array getAttributes* () inherited from *Phalcon\Assets\Resource*

Returns extra HTML attributes set in the resource

public *Phalcon\Assets\Resource* **setTargetUri** (*string \$targetUri*) inherited from *Phalcon\Assets\Resource*

Sets a target uri for the generated HTML

public *string getTargetUri* () inherited from *Phalcon\Assets\Resource*

Returns the target uri for the generated HTML

public *Phalcon\Assets\Resource* **setSourcePath** (*string \$sourcePath*) inherited from *Phalcon\Assets\Resource*

Sets the resource's source path

public *string getSourcePath* () inherited from *Phalcon\Assets\Resource*

Returns the resource's target path

public *Phalcon\Assets\Resource* **setTargetPath** (*string \$targetPath*) inherited from *Phalcon\Assets\Resource*

Sets the resource's target path

public *string getTargetPath* () inherited from *Phalcon\Assets\Resource*

Returns the resource's target path

public *string* **getContent** ([*string* \$basePath]) inherited from Phalcon\Assets\Resource

Returns the content of the resource as an string Optionally a base path where the resource is located can be set

public *string* **getRealTargetUri** () inherited from Phalcon\Assets\Resource

Returns the real target uri for the generated HTML

public *string* **getRealSourcePath** ([*string* \$basePath]) inherited from Phalcon\Assets\Resource

Returns the complete location where the resource is located

public *string* **getRealTargetPath** ([*string* \$basePath]) inherited from Phalcon\Assets\Resource

Returns the complete location where the resource must be written

## 2.54.26 Class Phalcon\CLI\Console

*implements Phalcon\DI\InjectionAwareInterface, Phalcon\Events\EventsAwareInterface*

This component allows to create CLI applications using Phalcon

### Methods

public **\_\_construct** ([*unknown* \$dependencyInjector])

Phalcon\CLI\Console constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the DependencyInjector container

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager)

Sets the events manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** ()

Returns the internal event manager

public **registerModules** (*array* \$modules)

Register an array of modules present in the console

<?php

```
$application->registerModules(array(
    'frontend' => array(
        'className' => 'Multiple\Frontend\Module',
        'path' => '../apps/frontend/Module.php'
    ),
    'backend' => array(
        'className' => 'Multiple\Backend\Module',
        'path' => '../apps/backend/Module.php'
    )
));
```

public **addModules** (*array* \$modules)

Merge modules with the existing ones

<?php

```
$application->addModules(array(
    'admin' => array(
        'className' => 'Multiple\Admin\Module',
        'path' => '../apps/admin/Module.php'
    )
));
```

public *array* **getModules** ()

Return the modules registered in the console

public *mixed* **handle** ([*array* \$arguments])

Handle the command-line arguments.

<?php

```
$arguments = array(
    'task' => 'taskname',
    'action' => 'action',
    'params' => array('parameter1', 'parameter2')
);
$console->handle($arguments);
```

## 2.54.27 Class Phalcon\CLI\Console\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\CLI\Console will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from *Exception*

Returns previous *Exception*

final public *Exception* **getTraceAsString** () inherited from *Exception*

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from *Exception*

String representation of the exception

## 2.54.28 Class Phalcon\CLI\Dispatcher

*extends* abstract class *Phalcon\Dispatcher*

*implements* *Phalcon\Events\EventsAwareInterface*, *Phalcon\DI\InjectionAwareInterface*, *Phalcon\DispatcherInterface*

Dispatching is the process of taking the command-line arguments, extracting the module name, task name, action name, and optional parameters contained in it, and then instantiating a task and calling an action on it.

```
<?php  
  
$di = new Phalcon\DI();  
  
$dispatcher = new Phalcon\CLI\Dispatcher();  
  
$dispatcher->setDI($di);  
  
$dispatcher->setTaskName('posts');  
$dispatcher->setActionName('index');  
$dispatcher->setParams(array());  
  
$handle = $dispatcher->dispatch();
```

### Constants

*integer* **EXCEPTION\_NO\_DI**

*integer* **EXCEPTION\_CYCLIC\_ROUTING**

*integer* **EXCEPTION\_HANDLER\_NOT\_FOUND**

*integer* **EXCEPTION\_INVALID\_HANDLER**

*integer* **EXCEPTION\_INVALID\_PARAMS**

*integer* **EXCEPTION\_ACTION\_NOT\_FOUND**

### Methods

public **setTaskSuffix** (*string* \$taskSuffix)

Sets the default task suffix

public **setDefaultTask** (*string* \$taskName)

Sets the default task name

public **setTaskName** (*string* \$taskName)

Sets the task name to be dispatched

public *string* **getTaskName** ()

Gets last dispatched task name

protected **\_throwDispatchException** ()

Throws an internal exception

protected **\_handleException** ()

Handles a user exception

public *string* **getTaskClass** ()

Possible task class name that will be located to dispatch the request

public *Phalcon|CLI|Task* **getLastTask** ()

Returns the lastest dispatched controller

public *Phalcon|CLI|Task* **getActiveTask** ()

Returns the active task in the dispatcher

public **\_\_construct** () inherited from Phalcon\Dispatcher

Phalcon\Dispatcher constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\Dispatcher

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\Dispatcher

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\Dispatcher

Sets the events manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\Dispatcher

Returns the internal event manager

public **setActionSuffix** (*string* \$actionSuffix) inherited from Phalcon\Dispatcher

Sets the default action suffix

public **setModuleName** (*string* \$moduleName) inherited from Phalcon\Dispatcher

Sets the module where the controller is (only informative)

public *string* **getModuleName** () inherited from Phalcon\Dispatcher

Gets the module where the controller class is

public **setNamespaceName** (*string* \$namespaceName) inherited from Phalcon\Dispatcher

Sets the namespace where the controller class is

public *string* **getNamespaceName** () inherited from Phalcon\Dispatcher

Gets a namespace to be prepended to the current handler name

public **setDefaultNamespace** (*string* \$namespace) inherited from Phalcon\Dispatcher

Sets the default namespace

public *string* **getDefaultNamespace** () inherited from Phalcon\Dispatcher

Returns the default namespace

public **setDefaultAction** (*string* \$actionName) inherited from Phalcon\Dispatcher

Sets the default action name

public **setActionName** (*string* \$actionName) inherited from Phalcon\Dispatcher

Sets the action name to be dispatched

public *string* **getActionName** () inherited from Phalcon\Dispatcher

Gets the lastest dispatched action name

public **setParams** (*array* \$params) inherited from Phalcon\Dispatcher

Sets action params to be dispatched

public *array* **getParams** () inherited from Phalcon\Dispatcher

Gets action params

public **setParam** (*mixed* \$param, *mixed* \$value) inherited from Phalcon\Dispatcher

Set a param by its name or numeric index

public *mixed* **getParam** (*mixed* \$param, [*string/array* \$filters]) inherited from Phalcon\Dispatcher

Gets a param by its name or numeric index

public *string* **getActiveMethod** () inherited from Phalcon\Dispatcher

Returns the current method to be/executed in the dispatcher

public *boolean* **isFinished** () inherited from Phalcon\Dispatcher

Checks if the dispatch loop is finished or has more pendent controllers/tasks to disptach

public **setReturnedValue** (*mixed* \$value) inherited from Phalcon\Dispatcher

Sets the latest returned value by an action manually

public *mixed* **getReturnedValue** () inherited from Phalcon\Dispatcher

Returns value returned by the lastest dispatched action

public *object* **dispatch** () inherited from Phalcon\Dispatcher

Dispatches a handle action taking into account the routing parameters

public **forward** (*array* \$forward) inherited from Phalcon\Dispatcher

Forwards the execution flow to another controller/action Dispatchers are unique per module. Forwarding between modules is not allowed

<?php

```
$this->dispatcher->forward(array('controller' => 'posts', 'action' => 'index'));
```

public *boolean* **wasForwarded** () inherited from Phalcon\Dispatcher

Check if the current executed action was forwarded by another one

public *string* **getHandlerClass** () inherited from Phalcon\Dispatcher

Possible class name that will be located to dispatch the request

## 2.54.29 Class Phalcon\CLI\Dispatcher\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\CLI\Dispatcher will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.30 Class Phalcon\CLI\Router

*implements Phalcon\DI\InjectionAwareInterface*

Phalcon\CLI\Router is the standard framework router. Routing is the process of taking a command-line arguments and decomposing it into parameters to determine which module, task, and action of that task should receive the request

<?php

```
$router = new Phalcon\CLI\Router();
$router->handle(array(
    'module' => 'main',
```

```
'task' => 'videos',
'action' => 'process'
));
echo $router->getTaskName();
```

## Methods

public **\_\_construct** ()

Phalcon\CLI\Router constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

public **setDefaultModule** (*string* \$moduleName)

Sets the name of the default module

public **setDefaultTask** (*string* \$taskId)

Sets the default controller name

public **setDefaultAction** (*string* \$actionName)

Sets the default action name

public **handle** ([*array* \$arguments])

Handles routing information received from command-line arguments

public *string* **getModuleName** ()

Returns proccesed module name

public *string* **getTaskName** ()

Returns proccesed task name

public *string* **getActionName** ()

Returns proccesed action name

public *array* **getParams** ()

Returns proccesed extra params

### 2.54.31 Class Phalcon\CLI\Router\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\CLI\Router will use this class

## Methods

final private *Exception* **`__clone()`** inherited from *Exception*  
Clone the exception

public **`__construct ([string $message], [int $code], [Exception $previous])`** inherited from *Exception*  
*Exception* constructor

final public *string* **`getMessage()`** inherited from *Exception*  
Gets the *Exception* message

final public *int* **`getCode()`** inherited from *Exception*  
Gets the *Exception* code

final public *string* **`getFile()`** inherited from *Exception*  
Gets the file in which the exception occurred

final public *int* **`getLine()`** inherited from *Exception*  
Gets the line in which the exception occurred

final public *array* **`getTrace()`** inherited from *Exception*  
Gets the stack trace

final public *Exception* **`getPrevious()`** inherited from *Exception*  
Returns previous *Exception*

final public *Exception* **`getTraceAsString()`** inherited from *Exception*  
Gets the stack trace as a string

public *string* **`__toString()`** inherited from *Exception*  
String representation of the exception

### 2.54.32 Class Phalcon\CLI\Task

*extends* abstract class *Phalcon\DI\Injectable*

*implements* *Phalcon\Events\EventsAwareInterface*, *Phalcon\DI\InjectionAwareInterface*

Every command-line task should extend this class that encapsulates all the task functionality A task can be used to run “tasks” such as migrations, cronjobs, unit-tests, or anything that you want. The Task class should at least have a “mainAction” method

```
<?php

class HelloTask extends \Phalcon\CLI\Task
{

    //This action will be executed by default
    public function mainAction()
    {

    }

    public function findAction()
    {
}
```

```
    }  
}
```

## Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\DI\Injectable

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\DI\Injectable

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\DI\Injectable

Returns the internal event manager

public **\_\_get** (*unknown* \$property) inherited from Phalcon\DI\Injectable

Magic method \_\_get

### 2.54.33 Abstract class Phalcon\Cache\Backend

*implements Phalcon\Cache\BackendInterface*

This class implements common functionality for backend adapters. A backend cache adapter may extend this class

## Methods

public **\_\_construct** (*Phalcon\Cache\FrontendInterface* \$frontend, [*array* \$options])

Phalcon\Cache\Backend constructor

public *mixed* **start** (*int/string* \$keyName, [*long* \$lifetime])

Starts a cache. The \$keyname allows to identify the created fragment

public **stop** ([*boolean* \$stopBuffer])

Stops the frontend without store any cached content

public *mixed* **getFrontend** ()

Returns front-end instance adapter related to the back-end

public *array* **getOptions** ()

Returns the backend options

public *boolean* **isFresh** ()

Checks whether the last cache is fresh or cached

public *boolean* **isStarted** ()

Checks whether the cache has starting buffering or not

`public setLastKey (string $lastKey)`

Sets the last key used in the cache

`public string getLastKey ()`

Gets the last key stored by the cache

`public int getLifetime ()`

Gets the last lifetime set

`abstract public mixed get (int/string $keyName, [long $lifetime]) inherited from Phalcon\Cache\BackendInterface`

Returns a cached content

`abstract public save ([int/string $keyName], [string $content], [long $lifetime], [boolean $stopBuffer]) inherited from Phalcon\Cache\BackendInterface`

Stores cached content into the file backend and stops the frontend

`abstract public boolean delete (int/string $keyName) inherited from Phalcon\Cache\BackendInterface`

Deletes a value from the cache by its key

`abstract public array queryKeys ([string $prefix]) inherited from Phalcon\Cache\BackendInterface`

Query the existing cached keys

`abstract public boolean exists ([string $keyName], [long $lifetime]) inherited from Phalcon\Cache\BackendInterface`

Checks if cache exists and it hasn't expired

`abstract public boolean flush () inherited from Phalcon\Cache\BackendInterface`

Immediately invalidates all existing items.

### 2.54.34 Class Phalcon\Cache\Backend\Apc

*extends abstract class Phalcon\Cache\Backend*

*implements Phalcon\Cache\BackendInterface*

Allows to cache output fragments, PHP data and raw data using an APC backend

`<?php`

```
//Cache data for 2 days
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    'lifetime' => 172800
));

$cache = new Phalcon\Cache\Backend\Apc($frontCache, array(
    'prefix' => 'app-data'
));

//Cache arbitrary data
$cache->save('my-data', array(1, 2, 3, 4, 5));

//Get data
$data = $cache->get('my-data');
```

## Methods

public *mixed* **get** (*string* \$keyName, [*long* \$lifetime])

Returns a cached content

public **save** ([*string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])

Stores cached content into the APC backend and stops the frontend

public *boolean* **delete** (*string* \$keyName)

Deletes a value from the cache by its key

public *array* **queryKeys** ([*string* \$prefix])

Query the existing cached keys

public *boolean* **exists** ([*string* \$keyName], [*long* \$lifetime])

Checks if cache exists and it hasn't expired

public *mixed* **increment** ([*unknown* \$key\_name], [*long* \$value])

Increment of a given key, by number \$value

public *mixed* **decrement** ([*unknown* \$key\_name], [*long* \$value])

Decrement of a given key, by number \$value

public *boolean* **flush** ()

Immediately invalidates all existing items.

public **\_\_construct** (*Phalcon\Cache\FrontendInterface* \$frontend, [*array* \$options]) inherited from Phalcon\Cache\Backend

Phalcon\Cache\Backend constructor

public *mixed* **start** (*int/string* \$keyName, [*long* \$lifetime]) inherited from Phalcon\Cache\Backend

Starts a cache. The \$keyname allows to identify the created fragment

public **stop** ([*boolean* \$stopBuffer]) inherited from Phalcon\Cache\Backend

Stops the frontend without store any cached content

public *mixed* **getFrontend** () inherited from Phalcon\Cache\Backend

Returns front-end instance adapter related to the back-end

public *array* **getOptions** () inherited from Phalcon\Cache\Backend

Returns the backend options

public *boolean* **isFresh** () inherited from Phalcon\Cache\Backend

Checks whether the last cache is fresh or cached

public *boolean* **isStarted** () inherited from Phalcon\Cache\Backend

Checks whether the cache has starting buffering or not

public **setLastKey** (*string* \$lastKey) inherited from Phalcon\Cache\Backend

Sets the last key used in the cache

`public string getLastKey ()` inherited from `Phalcon\Cache\Backend`

Gets the last key stored by the cache

`public int getLifetime ()` inherited from `Phalcon\Cache\Backend`

Gets the last lifetime set

### 2.54.35 Class `Phalcon\Cache\Backend\File`

*extends abstract class `Phalcon\Cache\Backend`*

*implements `Phalcon\Cache\BackendInterface`*

Allows to cache output fragments using a file backend

`<?php`

```
//Cache the file for 2 days
$frontendOptions = array(
    'lifetime' => 172800
);

//Create a output cache
$frontCache = \Phalcon\Cache\Frontend\Output($frontendOptions);

//Set the cache directory
$backendOptions = array(
    'cacheDir' => '../app/cache/',
);

//Create the File backend
$cache = new \Phalcon\Cache\Backend\File($frontCache, $backendOptions);

$content = $cache->start('my-cache');
if ($content === null) {
    echo '<h1>', time(), '</h1>';
    $cache->save();
} else {
    echo $content;
}
```

#### Methods

`public __construct (Phalcon\Cache\FrontendInterface $frontend, [array $options])`

`Phalcon\Cache\Backend\File` constructor

`public mixed get (int/string $keyName, [long $lifetime])`

Returns a cached content

`public save ([int/string $keyName], [string $content], [long $lifetime], [boolean $stopBuffer])`

Stores cached content into the file backend and stops the frontend

`public boolean delete (int/string $keyName)`

Deletes a value from the cache by its key

`public array queryKeys ([string $prefix])`

Query the existing cached keys

`public boolean exists ([string $keyName], [long $lifetime])`

Checks if cache exists and it isn't expired

`public mixed increment ([unknown $key_name], [long $value])`

Increment of a given key, by number \$value

`public mixed decrement ([unknown $key_name], [long $value])`

Decrement of a given key, by number \$value

`public boolean flush ()`

Immediately invalidates all existing items.

`public mixed start (int/string $keyName, [long $lifetime])` inherited from Phalcon\Cache\Backend

Starts a cache. The \$keyname allows to identify the created fragment

`public stop ([boolean $stopBuffer])` inherited from Phalcon\Cache\Backend

Stops the frontend without store any cached content

`public mixed getFrontend ()` inherited from Phalcon\Cache\Backend

Returns front-end instance adapter related to the back-end

`public array getOptions ()` inherited from Phalcon\Cache\Backend

Returns the backend options

`public boolean isFresh ()` inherited from Phalcon\Cache\Backend

Checks whether the last cache is fresh or cached

`public boolean isStarted ()` inherited from Phalcon\Cache\Backend

Checks whether the cache has starting buffering or not

`public setLastKey (string $lastKey)` inherited from Phalcon\Cache\Backend

Sets the last key used in the cache

`public string getLastKey ()` inherited from Phalcon\Cache\Backend

Gets the last key stored by the cache

`public int getLifetime ()` inherited from Phalcon\Cache\Backend

Gets the last lifetime set

## 2.54.36 Class Phalcon\Cache\Backend\Libmemcached

*extends abstract class Phalcon\Cache\Backend*

*implements Phalcon\Cache\BackendInterface*

Allows to cache output fragments, PHP data or raw data to a libmemcached backend This adapter uses the special memcached key “\_PHCM” to store all the keys internally used by the adapter

`<?php`

```
// Cache data for 2 days
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 172800
```

```

));;

//Create the Cache setting memcached connection options
$cache = new Phalcon\Cache\Backend\Libmemcached($frontCache, array(
    'servers' => array(
        array('host' => 'localhost',
              'port' => 11211,
              'weight' => 1),
    ),
    'client' => array(
        Memcached::OPT_HASH => Memcached::HASH_MD5,
        Memcached::OPT_PREFIX_KEY => 'prefix.',
    )
));
//Cache arbitrary data
$cache->save('my-data', array(1, 2, 3, 4, 5));

//Get data
$data = $cache->get('my-data');

```

## Methods

**public \_\_construct (*Phalcon\Cache\FrontendInterface* \$frontend, [*array* \$options])**

Phalcon\Cache\Backend\Libmemcached constructor

**protected \_connect ()**

Create internal connection to memcached

**public mixed get (*int/string* \$keyName, [*long* \$lifetime])**

Returns a cached content

**public save ([*int/string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])**

Stores cached content into the Memcached backend and stops the frontend

**public boolean delete (*int/string* \$keyName)**

Deletes a value from the cache by its key

**public array queryKeys ([*string* \$prefix])**

Query the existing cached keys

**public boolean exists ([*string* \$keyName], [*long* \$lifetime])**

Checks if cache exists and it hasn't expired

**public mixed increment ([*unknown* \$key\_name], [*long* \$value])**

Increment of a given key, by number \$value

**public mixed decrement ([*unknown* \$key\_name], [*long* \$value])**

Decrement of a given key, by number \$value

**public boolean flush ()**

Immediately invalidates all existing items.

**public getTrackingKey ()**

...

public **setTrackingKey** (*unknown \$key*)

...

public *mixed start* (*int/string \$keyName, [long \$lifetime]*) inherited from Phalcon\Cache\Backend  
Starts a cache. The \$keyname allows to identify the created fragment

public **stop** ([*boolean \$stopBuffer*]) inherited from Phalcon\Cache\Backend  
Stops the frontend without store any cached content

public *mixed getFrontend* () inherited from Phalcon\Cache\Backend  
Returns front-end instance adapter related to the back-end

public *array getOptions* () inherited from Phalcon\Cache\Backend  
Returns the backend options

public *boolean isFresh* () inherited from Phalcon\Cache\Backend  
Checks whether the last cache is fresh or cached

public *boolean isStarted* () inherited from Phalcon\Cache\Backend  
Checks whether the cache has starting buffering or not

public **setLastKey** (*string \$lastKey*) inherited from Phalcon\Cache\Backend  
Sets the last key used in the cache

public *string getLastKey* () inherited from Phalcon\Cache\Backend  
Gets the last key stored by the cache

public *int getLifetime* () inherited from Phalcon\Cache\Backend  
Gets the last lifetime set

### 2.54.37 Class Phalcon\Cache\Backend\Memcache

*extends abstract class Phalcon\Cache\Backend*

*implements Phalcon\Cache\BackendInterface*

Allows to cache output fragments, PHP data or raw data to a memcache backend This adapter uses the special memcached key “\_PHCM” to store all the keys internally used by the adapter

```
<?php

// Cache data for 2 days
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 172800
));

//Create the Cache setting memcached connection options
$cache = new Phalcon\Cache\Backend\Memcache($frontCache, array(
    'host' => 'localhost',
    'port' => 11211,
    'persistent' => false
));
```

```
//Cache arbitrary data
$cache->save('my-data', array(1, 2, 3, 4, 5));

//Get data
$data = $cache->get('my-data');
```

## Methods

**public \_\_construct (*Phalcon\Cache\FrontendInterface* \$frontend, [*array* \$options])**

Phalcon\Cache\Backend\Memcache constructor

**protected \_connect ()**

Create internal connection to memcached

**public mixed get (*int/string* \$keyName, [*long* \$lifetime])**

Returns a cached content

**public save ([*int/string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])**

Stores cached content into the Memcached backend and stops the frontend

**public boolean delete (*int/string* \$keyName)**

Deletes a value from the cache by its key

**public array queryKeys ([*string* \$prefix])**

Query the existing cached keys

**public boolean exists (*string* \$keyName, [*long* \$lifetime])**

Checks if cache exists and it hasn't expired

**public mixed increment ([*unknown* \$key\_name], [*long* \$value])**

Atomic increment of a given key, by number \$value

**public mixed decrement ([*unknown* \$key\_name], [*long* \$value])**

Atomic decrement of a given key, by number \$value

**public boolean flush ()**

Immediately invalidates all existing items.

**public getTrackingKey ()**

...

**public setTrackingKey (*unknown* \$key)**

...

**public mixed start (*int/string* \$keyName, [*long* \$lifetime])** inherited from Phalcon\Cache\Backend

Starts a cache. The \$keyname allows to identify the created fragment

**public stop ([*boolean* \$stopBuffer])** inherited from Phalcon\Cache\Backend

Stops the frontend without store any cached content

**public mixed getFrontend ()** inherited from Phalcon\Cache\Backend

Returns front-end instance adapter related to the back-end

public *array* **getOptions** () inherited from Phalcon\Cache\Backend  
Returns the backend options

public *boolean* **isFresh** () inherited from Phalcon\Cache\Backend  
Checks whether the last cache is fresh or cached

public *boolean* **isStarted** () inherited from Phalcon\Cache\Backend  
Checks whether the cache has starting buffering or not

public **setLastKey** (*string* \$lastKey) inherited from Phalcon\Cache\Backend  
Sets the last key used in the cache

public *string* **getLastKey** () inherited from Phalcon\Cache\Backend  
Gets the last key stored by the cache

public *int* **getLifetime** () inherited from Phalcon\Cache\Backend  
Gets the last lifetime set

### 2.54.38 Class Phalcon\Cache\Backend\Memory

*extends* abstract class Phalcon\Cache\Backend

*implements* Phalcon\Cache\BackendInterface

Stores content in memory. Data is lost when the request is finished

```
<?php

//Cache data
$frontCache = new Phalcon\Cache\Frontend\Data();

$cache = new Phalcon\Cache\Backend\Memory($frontCache);

//Cache arbitrary data
$cache->save('my-data', array(1, 2, 3, 4, 5));

//Get data
$data = $cache->get('my-data');
```

#### Methods

public *mixed* **get** (*string* \$keyName, [*long* \$lifetime])

Returns a cached content

public **save** ([*string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])

Stores cached content into the backend and stops the frontend

public *boolean* **delete** (*string* \$keyName)

Deletes a value from the cache by its key

public *array* **queryKeys** ([*string* \$prefix])

Query the existing cached keys

public *boolean* **exists** ([*string* \$keyName], [*long* \$lifetime])

Checks if cache exists and it hasn't expired

`public mixed increment ([unknown $key_name], [unknown $value])`

Increment of given \$keyName by \$value

`public long decrement ([unknown $key_name], [long $value])`

Decrement of \$keyName by given \$value

`public boolean flush ()`

Immediately invalidates all existing items.

`public __construct (Phalcon\Cache\FrontendInterface $frontend, [array $options])` inherited from Phalcon\Cache\Backend

Phalcon\Cache\Backend constructor

`public mixed start (int/string $keyName, [long $lifetime])` inherited from Phalcon\Cache\Backend

Starts a cache. The \$keyname allows to identify the created fragment

`public stop ([boolean $stopBuffer])` inherited from Phalcon\Cache\Backend

Stops the frontend without store any cached content

`public mixed getFrontend ()` inherited from Phalcon\Cache\Backend

Returns front-end instance adapter related to the back-end

`public array getOptions ()` inherited from Phalcon\Cache\Backend

Returns the backend options

`public boolean isFresh ()` inherited from Phalcon\Cache\Backend

Checks whether the last cache is fresh or cached

`public boolean isStarted ()` inherited from Phalcon\Cache\Backend

Checks whether the cache has starting buffering or not

`public setLastKey (string $lastKey)` inherited from Phalcon\Cache\Backend

Sets the last key used in the cache

`public string getLastKey ()` inherited from Phalcon\Cache\Backend

Gets the last key stored by the cache

`public int getLifetime ()` inherited from Phalcon\Cache\Backend

Gets the last lifetime set

### 2.54.39 Class Phalcon\Cache\Backend\Mongo

`extends abstract class Phalcon\Cache\Backend`

`implements Phalcon\Cache\BackendInterface`

Allows to cache output fragments, PHP data or raw data to a MongoDb backend

`<?php`

```
// Cache data for 2 days
$frontCache = new Phalcon\Cache\Frontend\Base64(array(
    "lifetime" => 172800
```

```
));

//Create a MongoDB cache
$cache = new Phalcon\Cache\Backend\Mongo($frontCache, array(
    'server' => "mongodb://localhost",
    'db' => 'caches',
    'collection' => 'images',
));

//Cache arbitrary data
$cache->save('my-data', file_get_contents('some-image.jpg'));

//Get data
$data = $cache->get('my-data');
```

## Methods

public **\_\_construct** (*Phalcon|Cache|FrontendInterface* \$frontend, [*array* \$options])

Phalcon\Cache\Backend\Mongo constructor

protected *MongoCollection* **\_getCollection** ()

Returns a MongoDB collection based on the backend parameters

public *mixed* **get** (*int/string* \$keyName, [*long* \$lifetime])

Returns a cached content

public **save** ([*int/string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])

Stores cached content into the Mongo backend and stops the frontend

public **boolean** **delete** (*int/string* \$keyName)

Deletes a value from the cache by its key

public *array* **queryKeys** ([*string* \$prefix])

Query the existing cached keys

public **boolean** **exists** ([*string* \$keyName], [*long* \$lifetime])

Checks if cache exists and it hasn't expired

public **gc** ()

...

public *mixed* **increment** ([*unknown* \$key\_name], [*long* \$value])

Increment of a given key by \$value

public *mixed* **decrement** ([*unknown* \$key\_name], [*long* \$value])

Decrement of a given key by \$value

public **bool** **flush** ()

Immediately invalidates all existing items.

public *mixed* **start** (*int/string* \$keyName, [*long* \$lifetime]) inherited from Phalcon\Cache\Backend

Starts a cache. The \$keyname allows to identify the created fragment

public **stop** ([*boolean* \$stopBuffer]) inherited from Phalcon\Cache\Backend

Stops the frontend without store any cached content

`public mixed getFrontend ()` inherited from `Phalcon\Cache\Backend`

Returns front-end instance adapter related to the back-end

`public array getOptions ()` inherited from `Phalcon\Cache\Backend`

Returns the backend options

`public boolean isFresh ()` inherited from `Phalcon\Cache\Backend`

Checks whether the last cache is fresh or cached

`public boolean isStarted ()` inherited from `Phalcon\Cache\Backend`

Checks whether the cache has starting buffering or not

`public setLastKey (string $lastKey)` inherited from `Phalcon\Cache\Backend`

Sets the last key used in the cache

`public string getLastKey ()` inherited from `Phalcon\Cache\Backend`

Gets the last key stored by the cache

`public int getLifetime ()` inherited from `Phalcon\Cache\Backend`

Gets the last lifetime set

#### 2.54.40 Class `Phalcon\Cache\Backend\Xcache`

*extends abstract class `Phalcon\Cache\Backend`*

*implements `Phalcon\Cache\BackendInterface`*

Allows to cache output fragments, PHP data and raw data using an XCache backend

`<?php`

```
//Cache data for 2 days
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    'lifetime' => 172800
));

$cache = new Phalcon\Cache\Backend\Xcache($frontCache, array(
    'prefix' => 'app-data'
));

//Cache arbitrary data
$cache->save('my-data', array(1, 2, 3, 4, 5));

//Get data
$data = $cache->get('my-data');
```

#### Methods

`public __construct (Phalcon\Cache\FrontendInterface $frontend, [array $options])`

Phalcon\Cache\Backend\Xcache constructor

`public mixed get (string $keyName, [long $lifetime])`

Returns cached content

`public save ([string $keyName], [string $content], [long $lifetime], [boolean $stopBuffer])`

Stores cached content into the XCache backend and stops the frontend

`public boolean delete (string $keyName)`

Deletes a value from the cache by its key

`public array queryKeys ([string $prefix])`

Query the existing cached keys

`public boolean exists ([string $keyName], [long $lifetime])`

Checks if the cache entry exists and has not expired

`public mixed increment ([unknown $key_name], [long $value])`

Atomic increment of a given key, by number \$value

`public mixed decrement ([unknown $key_name], [long $value])`

Atomic decrement of a given key, by number \$value

`public boolean flush ()`

Immediately invalidates all existing items.

`public mixed start (int/string $keyName, [long $lifetime])` inherited from Phalcon\Cache\Backend

Starts a cache. The \$keyname allows to identify the created fragment

`public stop ([boolean $stopBuffer])` inherited from Phalcon\Cache\Backend

Stops the frontend without store any cached content

`public mixed getFrontend ()` inherited from Phalcon\Cache\Backend

Returns front-end instance adapter related to the back-end

`public array getOptions ()` inherited from Phalcon\Cache\Backend

Returns the backend options

`public boolean isFresh ()` inherited from Phalcon\Cache\Backend

Checks whether the last cache is fresh or cached

`public boolean isStarted ()` inherited from Phalcon\Cache\Backend

Checks whether the cache has starting buffering or not

`public setLastKey (string $lastKey)` inherited from Phalcon\Cache\Backend

Sets the last key used in the cache

`public string getLastKey ()` inherited from Phalcon\Cache\Backend

Gets the last key stored by the cache

`public int getLifetime ()` inherited from Phalcon\Cache\Backend

Gets the last lifetime set

## 2.54.41 Class Phalcon\Cache\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Cache will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.42 Class Phalcon\Cache\Frontend\Base64

*extends class Phalcon\Cache\Frontend\Data*

*implements Phalcon\Cache\FrontendInterface*

Allows to cache data converting/deconverting them to base64. This adapters uses the base64\_encode/base64\_decode PHP's functions

<?php

```
// Cache the files for 2 days using a Base64 frontend
$frontCache = new Phalcon\Cache\Frontend\Base64(array(
    "lifetime" => 172800
));
```

```
//Create a MongoDB cache
$cache = new Phalcon\Cache\Backend\Mongo($frontCache, array(
    'server' => "mongodb://localhost",
    'db' => 'caches',
    'collection' => 'images',
));

// Try to get cached image
$cacheKey = 'some-image.jpg.cache';
$image    = $cache->get($cacheKey);
if ($image === null) {

    // Store the image in the cache
    $cache->save($cacheKey, file_get_contents('tmp-dir/some-image.jpg'));
}

header('Content-Type: image/jpeg');
echo $image;
```

## Methods

public *string* **beforeStore** (*mixed* \$data)

Serializes data before storing them

public *mixed* **afterRetrieve** (*mixed* \$data)

Unserializes data after retrieval

public **\_\_construct** ([*array* \$frontendOptions]) inherited from Phalcon\Cache\Frontend\Data

Phalcon\Cache\Frontend\Data constructor

public *int* **getLifetime** () inherited from Phalcon\Cache\Frontend\Data

Returns cache lifetime

public *boolean* **isBuffering** () inherited from Phalcon\Cache\Frontend\Data

Check whether if frontend is buffering output

public **start** () inherited from Phalcon\Cache\Frontend\Data

Starts output frontend. Actually, does nothing

public *string* **getContent** () inherited from Phalcon\Cache\Frontend\Data

Returns output cached content

public **stop** () inherited from Phalcon\Cache\Frontend\Data

Stops output frontend

### 2.54.43 Class Phalcon\Cache\Frontend\Data

*implements* Phalcon\Cache\FrontendInterface

Allows to cache native PHP data in a serialized form

```
<?php

// Cache the files for 2 days using a Data frontend
$frontCache = new Phalcon\Cache\Frontend\Data(array(
    "lifetime" => 172800
));

// Create the component that will cache "Data" to a "File" backend
// Set the cache file directory - important to keep the "/" at the end of
// of the value for the folder
$cache = new Phalcon\Cache\Backend\File($frontCache, array(
    "cacheDir" => "../app/cache/"
));

// Try to get cached records
$cacheKey = 'robots_order_id.cache';
$robots = $cache->get($cacheKey);
if ($robots === null) {

    // $robots is null due to cache expiration or data does not exist
    // Make the database call and populate the variable
    $robots = Robots::find(array("order" => "id"));

    // Store it in the cache
    $cache->save($cacheKey, $robots);
}

// Use $robots :)
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}
```

## Methods

public **\_\_construct** ([array \$frontendOptions])

Phalcon\Cache\Frontend\Data constructor

public **int getLifetime** ()

Returns cache lifetime

public **boolean isBuffering** ()

Check whether if frontend is buffering output

public **start** ()

Starts output frontend. Actually, does nothing

public **string getContent** ()

Returns output cached content

public **stop** ()

Stops output frontend

public **string beforeStore** (*mixed* \$data)

Serializes data before storing them

public *mixed* **afterRetrieve** (*mixed* \$data)

Unserializes data after retrieval

## 2.54.44 Class Phalcon\Cache\Frontend\Igbinary

*extends class Phalcon\Cache\Frontend\Data*

*implements Phalcon\Cache\FrontendInterface*

Allows to cache native PHP data in a serialized form using igbinary extension

<?php

```
// Cache the files for 2 days using Igbinary frontend
$frontCache = new Phalcon\Cache\Frontend\Igbinary(array(
    "lifetime" => 172800
));

// Create the component that will cache "Igbinary" to a "File" backend
// Set the cache file directory - important to keep the "/" at the end of
// of the value for the folder
$cache = new Phalcon\Cache\Backend\File($frontCache, array(
    "cacheDir" => "../app/cache/"
));

// Try to get cached records
$cacheKey   = 'robots_order_id.cache';
$robots     = $cache->get($cacheKey);
if ($robots === null) {

    // $robots is null due to cache expiration or data do not exist
    // Make the database call and populate the variable
    $robots = Robots::find(array("order" => "id"));

    // Store it in the cache
    $cache->save($cacheKey, $robots);
}

// Use $robots :)
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}
```

### Methods

public *string* **beforeStore** (*mixed* \$data)

Serializes data before storing them

public *mixed* **afterRetrieve** (*mixed* \$data)

Unserializes data after retrieval

public **\_\_construct** ([array \$frontendOptions]) inherited from Phalcon\Cache\Frontend\Data

Phalcon\Cache\Frontend\Data constructor

public *int* **getLifetime** () inherited from Phalcon\Cache\Frontend\Data

Returns cache lifetime

`public boolean isBuffering ()` inherited from `Phalcon\Cache\Frontend\Data`

Check whether if frontend is buffering output

`public start ()` inherited from `Phalcon\Cache\Frontend\Data`

Starts output frontend. Actually, does nothing

`public string getContent ()` inherited from `Phalcon\Cache\Frontend\Data`

Returns output cached content

`public stop ()` inherited from `Phalcon\Cache\Frontend\Data`

Stops output frontend

## 2.54.45 Class `Phalcon\Cache\Frontend\Json`

*extends class `Phalcon\Cache\Frontend\Data`*

*implements `Phalcon\Cache\FrontendInterface`*

Allows to cache data converting/deconverting them to JSON. This adapters uses the `json_encode/json_decode` PHP's functions As the data is encoded in JSON other systems accessing the same backend could process them

`<?php`

```
// Cache the data for 2 days
$frontCache = new Phalcon\Cache\Frontend\Json(array(
    "lifetime" => 172800
));

//Create the Cache setting memcached connection options
$cache = new Phalcon\Cache\Backend\Memcache($frontCache, array(
    'host' => 'localhost',
    'port' => 11211,
    'persistent' => false
));

//Cache arbitrary data
$cache->save('my-data', array(1, 2, 3, 4, 5));

//Get data
$data = $cache->get('my-data');
```

### Methods

`public string beforeStore (mixed $data)`

Serializes data before storing it

`public mixed afterRetrieve (mixed $data)`

Unserializes data after retrieving it

`public __construct ([array $frontendOptions])` inherited from `Phalcon\Cache\Frontend\Data`

`Phalcon\Cache\Frontend\Data` constructor

public *int* **getLifetime** () inherited from Phalcon\Cache\Frontend\Data

Returns cache lifetime

public *boolean* **isBuffering** () inherited from Phalcon\Cache\Frontend\Data

Check whether if frontend is buffering output

public **start** () inherited from Phalcon\Cache\Frontend\Data

Starts output frontend. Actually, does nothing

public *string* **getContent** () inherited from Phalcon\Cache\Frontend\Data

Returns output cached content

public **stop** () inherited from Phalcon\Cache\Frontend>Data

Stops output frontend

## 2.54.46 Class Phalcon\Cache\Frontend\None

*extends class Phalcon\Cache\Frontend\Data*

*implements Phalcon\Cache\FrontendInterface*

Discards any kind of frontend data input. This frontend does not have expiration time or any other options

<?php

```
//Create a None Cache
$frontCache = new Phalcon\Cache\Frontend\None();

// Create the component that will cache "Data" to a "Memcached" backend
// Memcached connection settings
$cache = new Phalcon\Cache\Backend\Memcache($frontCache, array(
    "host" => "localhost",
    "port" => "11211"
));

// This Frontend always return the data as it's returned by the backend
$cacheKey = 'robots_order_id.cache';
$robots   = $cache->get($cacheKey);
if ($robots === null) {

    // This cache doesn't perform any expiration checking, so the data is always expired
    // Make the database call and populate the variable
    $robots = Robots::find(array("order" => "id"));

    $cache->save($cacheKey, $robots);
}

// Use $robots :
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}
```

### Methods

public *int* **getLifetime** ()

Returns cache lifetime, always one second expiring content

**public beforeStore (mixed \$data)**

Prepare data to be stored

**public afterRetrieve (mixed \$data)**

Prepares data to be retrieved to user

**public \_\_construct ([array \$frontendOptions])** inherited from Phalcon\Cache\Frontend\Data

Phalcon\Cache\Frontend\Data constructor

**public boolean isBuffering ()** inherited from Phalcon\Cache\Frontend\Data

Check whether if frontend is buffering output

**public start ()** inherited from Phalcon\Cache\Frontend\Data

Starts output frontend. Actually, does nothing

**public string getContent ()** inherited from Phalcon\Cache\Frontend\Data

Returns output cached content

**public stop ()** inherited from Phalcon\Cache\Frontend\Data

Stops output frontend

## 2.54.47 Class Phalcon\Cache\Frontend\Output

*implements Phalcon\Cache\FrontendInterface*

Allows to cache output fragments captured with ob\_\* functions

<?php

```
//Create an Output frontend. Cache the files for 2 days
$frontCache = new Phalcon\Cache\Frontend\Output(array(
    "lifetime" => 172800
));

// Create the component that will cache from the "Output" to a "File" backend
// Set the cache file directory - it's important to keep the "/" at the end of
// the value for the folder
$cache = new Phalcon\Cache\Backend\File($frontCache, array(
    "cacheDir" => "../app/cache/"
));

// Get/Set the cache file to ../app/cache/my-cache.html
$content = $cache->start("my-cache.html");

// If $content is null then the content will be generated for the cache
if ($content === null) {

    //Print date and time
    echo date("r");

    //Generate a link to the sign-up action
    echo Phalcon\Tag::linkTo(
        array(
            "user/signup",

```

```
        "Sign Up",
        "class" => "signup-button"
    )
);

// Store the output into the cache file
$cache->save();

} else {

    // Echo the cached output
    echo $content;
}
```

## Methods

public **\_\_construct** ([array \$frontendOptions])

Phalcon\Cache\Frontend\Output constructor

public **getLifetime** ()

Returns cache lifetime

public **isBuffering** ()

Check whether if frontend is buffering output

public **start** ()

Starts output frontend

public **getContent** ()

Returns output cached content

public **stop** ()

Stops output frontend

public **mixed beforeStore** (*mixed* \$data)

Prepare data to be stored

public **mixed afterRetrieve** (*mixed* \$data)

Prepares data to be retrieved to user

### 2.54.48 Class Phalcon\Cache\Multiple

Allows to read to chained backends writing to multiple backends

<?php

```
use Phalcon\Cache\Frontend\Data as DataFrontend,
Phalcon\Cache\Multiple,
Phalcon\Cache\Backend\Apc as ApcCache,
Phalcon\Cache\Backend\Memcache as MemcacheCache,
Phalcon\Cache\Backend\File as FileCache;

$ultraFastFrontend = new DataFrontend(array(
```

```

        "lifetime" => 3600
    ));
}

$fastFrontend = new DataFrontend(array(
    "lifetime" => 86400
));

$slowFrontend = new DataFrontend(array(
    "lifetime" => 604800
));

//Backends are registered from the fastest to the slower
$cache = new Multiple(array(
    new ApcCache($ultraFastFrontend, array(
        "prefix" => 'cache',
    )),
    new MemcacheCache($fastFrontend, array(
        "prefix" => 'cache',
        "host" => "localhost",
        "port" => "11211"
    )),
    new FileCache($slowFrontend, array(
        "prefix" => 'cache',
        "cacheDir" => "../app/cache/"
    ))
));
$cache->save('my-key', $data);

```

## Methods

**public \_\_construct ([*Phalcon|Cache|BackendInterface*] \$backends)**  
*Phalcon\Cache\Multiple* constructor

**public *Phalcon|Cache|Multiple* push (*Phalcon|Cache|BackendInterface* \$backend)**  
 Adds a backend

**public *mixed* get (*string* \$keyName, [*long* \$lifetime])**  
 Returns a cached content reading the internal backends

**public *mixed* start (*int/string* \$keyName, [*long* \$lifetime])**  
 Starts every backend

**public *save* ([*string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])**  
 Stores cached content into all backends and stops the frontend

**public *boolean* delete (*int/string* \$keyName)**  
 Deletes a value from each backend

**public *boolean* exists ([*string* \$keyName], [*long* \$lifetime])**  
 Checks if cache exists in at least one backend

## 2.54.49 Class Phalcon\Config

*implements* ArrayAccess, Countable

Phalcon\Config is designed to simplify the access to, and the use of, configuration data within applications. It provides a nested object property based user interface for accessing this configuration data within application code.

<?php

```
$config = new Phalcon\Config(array(
    "database" => array(
        "adapter" => "Mysql",
        "host" => "localhost",
        "username" => "scott",
        "password" => "cheetah",
        "dbname" => "test_db"
    ),
    "phalcon" => array(
        "controllersDir" => "../app/controllers/",
        "modelsDir" => "../app/models/",
        "viewsDir" => "../app/views/"
    )
));
```

### Methods

public **\_\_construct** ([array \$arrayConfig])

Phalcon\Config constructor

public **boolean offsetExists** (*unknown* \$property)

Allows to check whether an attribute is defined using the array-syntax

<?php

```
var_dump(isset($config['database']));
```

public **mixed get** (*string* \$index, [*mixed* \$defaultValue])

Gets an attribute from the configuration, if the attribute isn't defined returns null If the value is exactly null or is not defined the default value will be used instead

<?php

```
echo $config->get('controllersDir', '../app/controllers/');
```

public **string offsetGet** (*unknown* \$property)

Gets an attribute using the array-syntax

<?php

```
print_r($config['database']);
```

public **offsetSet** (*unknown* \$property, *mixed* \$value)

Sets an attribute using the array-syntax

```
<?php

$config['database'] = array('type' => 'Sqlite');

public offsetUnset (unknown $property)
Unsets an attribute using the array-syntax

<?php

unset($config['database']);

public merge (Phalcon\Config $config)
Merges a configuration into the current one

<?php

$appConfig = new Phalcon\Config(array('database' => array('host' => 'localhost')));
$globalConfig->merge($config);

public array toArray ()
Converts recursively the object to an array

<?php

print_r($config->toArray());

public count ()
...

public __wakeup ()
...

public static Phalcon\Config __set_state ([unknown $properties])
Restores the state of a Phalcon\Config object

public __get (unknown $property)
...

public __set (unknown $property, unknown $value)
...

public __isset (unknown $property)
...

public __unset (unknown $property)
...
```

## 2.54.50 Class Phalcon\Config\Adapter\Ini

*extends class Phalcon\Config  
implements Countable, ArrayAccess*

Reads ini files and converts them to Phalcon\Config objects. Given the next configuration file:

```
<?php

[database]
adapter = Mysql
host = localhost
username = scott
password = cheetah
dbname = test_db

[phalcon]
controllersDir = "../app/controllers/"
modelsDir = "../app/models/"
viewsDir = "../app/views/"
```

You can read it as follows:

```
<?php

$config = new Phalcon\Config\Adapter\Ini("path/config.ini");
echo $config->phalcon->controllersDir;
echo $config->database->username;
```

## Methods

public **\_\_construct** (*string* \$filePath)

Phalcon\Config\Adapter\Ini constructor

public *boolean* **offsetExists** (*unknown* \$property) inherited from Phalcon\Config

Allows to check whether an attribute is defined using the array-syntax

```
<?php
```

```
var_dump(isset($config['database']));
```

public *mixed* **get** (*string* \$index, [*mixed* \$defaultValue]) inherited from Phalcon\Config

Gets an attribute from the configuration, if the attribute isn't defined returns null If the value is exactly null or is not defined the default value will be used instead

```
<?php
```

```
echo $config->get('controllersDir', '../app/controllers/');
```

public *string* **offsetGet** (*unknown* \$property) inherited from Phalcon\Config

Gets an attribute using the array-syntax

```
<?php
```

```
print_r($config['database']);
```

public **offsetSet** (*unknown* \$property, *mixed* \$value) inherited from Phalcon\Config

Sets an attribute using the array-syntax

```
<?php
```

```
$config['database'] = array('type' => 'Sqlite');
```

public **offsetUnset** (*unknown \$property*) inherited from Phalcon\Config

Unsets an attribute using the array-syntax

<?php

```
unset($config['database']);
```

public **merge** (*Phalcon\Config \$config*) inherited from Phalcon\Config

Merges a configuration into the current one

<?php

```
$appConfig = new Phalcon\Config(array('database' => array('host' => 'localhost')));  
$globalConfig->merge($config2);
```

public **array toArray** () inherited from Phalcon\Config

Converts recursively the object to an array

<?php

```
print_r($config->toArray());
```

public **count** () inherited from Phalcon\Config

...

public **\_\_wakeup** () inherited from Phalcon\Config

...

public static *Phalcon\Config \_\_set\_state ([unknown \$properties])* inherited from Phalcon\Config

Restores the state of a Phalcon\Config object

public **\_\_get** (*unknown \$property*) inherited from Phalcon\Config

...

public **\_\_set** (*unknown \$property, unknown \$value*) inherited from Phalcon\Config

...

public **\_\_isset** (*unknown \$property*) inherited from Phalcon\Config

...

public **\_\_unset** (*unknown \$property*) inherited from Phalcon\Config

...

## 2.54.51 Class Phalcon\Config\Adapter\Json

*extends class Phalcon\Config*

*implements Countable, ArrayAccess*

Reads JSON files and converts them to Phalcon\Config objects. Given the following configuration file:

<?php

```
{"phalcon":{"baseuri":"/phalcon/"},"models":{"metadata":"memory"}}
```

You can read it as follows:

```
<?php

$config = new Phalcon\Config\Adapter\Json("path/config.json");
echo $config->phalcon->baseuri;
echo $config->models->metadata;
```

## Methods

public **\_\_construct** (*string* \$filePath)

Phalcon\Config\Adapter\Json constructor

public **boolean offsetExists** (*unknown* \$property) inherited from Phalcon\Config

Allows to check whether an attribute is defined using the array-syntax

```
<?php
```

```
var_dump(isset($config['database']));
```

public **mixed get** (*string* \$index, [*mixed* \$defaultValue]) inherited from Phalcon\Config

Gets an attribute from the configuration, if the attribute isn't defined returns null If the value is exactly null or is not defined the default value will be used instead

```
<?php
```

```
echo $config->get('controllersDir', '../app/controllers/');
```

public **string offsetGet** (*unknown* \$property) inherited from Phalcon\Config

Gets an attribute using the array-syntax

```
<?php
```

```
print_r($config['database']);
```

public **offsetSet** (*unknown* \$property, *mixed* \$value) inherited from Phalcon\Config

Sets an attribute using the array-syntax

```
<?php
```

```
$config['database'] = array('type' => 'Sqlite');
```

public **offsetUnset** (*unknown* \$property) inherited from Phalcon\Config

Unsets an attribute using the array-syntax

```
<?php
```

```
unset($config['database']);
```

public **merge** (*Phalcon\Config* \$config) inherited from Phalcon\Config

Merges a configuration into the current one

```
<?php

$appConfig = new Phalcon\Config(array('database' => array('host' => 'localhost')));
$globalConfig->merge($config2);

public array toArray () inherited from Phalcon\Config
Converts recursively the object to an array

<?php

print_r($config->toArray());

public count () inherited from Phalcon\Config
...

public __wakeup () inherited from Phalcon\Config
...

public static Phalcon\Config __set_state ([unknown $properties]) inherited from Phalcon\Config
Restores the state of a Phalcon\Config object

public __get (unknown $property) inherited from Phalcon\Config
...

public __set (unknown $property, unknown $value) inherited from Phalcon\Config
...

public __isset (unknown $property) inherited from Phalcon\Config
...

public __unset (unknown $property) inherited from Phalcon\Config
...
```

## 2.54.52 Class Phalcon\Config\Adapter\Php

*extends class Phalcon\Config  
implements Countable, ArrayAccess*

Reads php files and converts them to Phalcon\Config objects. Given the next configuration file:

```
<?php
return array(
'database' => array(
'adapter' => 'Mysql',
'host' => 'localhost',
'username' => 'scott',
'password' => 'cheetah',
'dbname' => 'test_db'
),
'phalcon' => array(
'controllersDir' => '../app/controllers/>,
```

```
'modelsDir' => '../app/models/',
'vewsDir' => '../app/views/',
));
```

You can read it as follows:

```
<?php

$config = new Phalcon\Config\Adapter\Php("path/config.php");
echo $config->phalcon->controllersDir;
echo $config->database->username;
```

## Methods

public **\_\_construct** (*string* \$filePath)

Phalcon\Config\Adapter\Php constructor

public **boolean offsetExists** (*unknown* \$property) inherited from Phalcon\Config

Allows to check whether an attribute is defined using the array-syntax

```
<?php
```

```
var_dump(isset($config['database']));
```

public **mixed get** (*string* \$index, [*mixed* \$defaultValue]) inherited from Phalcon\Config

Gets an attribute from the configuration, if the attribute isn't defined returns null If the value is exactly null or is not defined the default value will be used instead

```
<?php
```

```
echo $config->get('controllersDir', '../app/controllers/');
```

public **string offsetGet** (*unknown* \$property) inherited from Phalcon\Config

Gets an attribute using the array-syntax

```
<?php
```

```
print_r($config['database']);
```

public **offsetSet** (*unknown* \$property, *mixed* \$value) inherited from Phalcon\Config

Sets an attribute using the array-syntax

```
<?php
```

```
$config['database'] = array('type' => 'Sqlite');
```

public **offsetUnset** (*unknown* \$property) inherited from Phalcon\Config

Unsets an attribute using the array-syntax

```
<?php
```

```
unset($config['database']);
```

public **merge** (*Phalcon\Config \$config*) inherited from Phalcon\Config

Merges a configuration into the current one

<?php

```
$appConfig = new Phalcon\Config(array('database' => array('host' => 'localhost')));
$globalConfig->merge($config2);
```

public *array toArray ()* inherited from Phalcon\Config

Converts recursively the object to an array

<?php

```
print_r($config->toArray());
```

public **count ()** inherited from Phalcon\Config

...

public **\_\_wakeup ()** inherited from Phalcon\Config

...

public static *Phalcon\Config \_\_set\_state ([unknown \$properties])* inherited from Phalcon\Config

Restores the state of a Phalcon\Config object

public **\_\_get (unknown \$property)** inherited from Phalcon\Config

...

public **\_\_set (unknown \$property, unknown \$value)** inherited from Phalcon\Config

...

public **\_\_isset (unknown \$property)** inherited from Phalcon\Config

...

public **\_\_unset (unknown \$property)** inherited from Phalcon\Config

...

## 2.54.53 Class Phalcon\Config\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Config will use this class

### Methods

final private *Exception \_\_clone ()* inherited from Exception

Clone the exception

public **\_\_construct ([string \$message], [int \$code], [Exception \$previous])** inherited from Exception

Exception constructor

final public *string getMessage ()* inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception  
Gets the Exception code

final public *string* **getFile** () inherited from Exception  
Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception  
Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception  
Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception  
Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception  
Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception  
String representation of the exception

## 2.54.54 Class Phalcon\Crypt

*implements Phalcon\ CryptInterface*

Provides encryption facilities to phalcon applications

```
<?php  
  
$crypt = new Phalcon\Crypt();  
  
$key = 'le password';  
$text = 'This is a secret text';  
  
$encrypted = $crypt->encrypt($text, $key);  
  
echo $crypt->decrypt($encrypted, $key);
```

### Constants

*integer* **PADDING\_DEFAULT**  
*integer* **PADDING\_ANSI\_X\_923**  
*integer* **PADDING\_PKCS7**  
*integer* **PADDING\_ISO\_10126**  
*integer* **PADDING\_ISO\_IEC\_7816\_4**  
*integer* **PADDING\_ZERO**  
*integer* **PADDING\_SPACE**

## Methods

public *Phalcon\Encrypt* **setCipher** (*string* \$cipher)

Sets the cipher algorithm

public *string* **getCipher** ()

Returns the current cipher

public *Phalcon\Encrypt* **setMode** (*unknown* \$mode)

Sets the encrypt/decrypt mode

public *string* **getMode** ()

Returns the current encryption mode

public *Phalcon\Encrypt* **setKey** (*string* \$key)

Sets the encryption key

public *string* **getKey** ()

Returns the encryption key

public *Phalcon\CryptInterface* **setPadding** (*unknown* \$scheme)

public *int* **getPadding** ()

Returns the padding scheme

public *string* **encrypt** (*string* \$text, [*string* \$key])

Encrypts a text

<?php

```
$encrypted = $crypt->encrypt("Ultra-secret text", "encrypt password");
```

public *string* **decrypt** (*string* \$text, [*string* \$key])

Decrypts an encrypted text

<?php

```
echo $crypt->decrypt($encrypted, "decrypt password");
```

public *string* **encryptBase64** (*string* \$text, [*string* \$key], [*unknown* \$safe])

Encrypts a text returning the result as a base64 string

public *string* **decryptBase64** (*string* \$text, [*string* \$key], [*unknown* \$safe])

Decrypt a text that is coded as a base64 string

public *array* **getAvailableCiphers** ()

Returns a list of available cyphers

public *array* **getAvailableModes** ()

Returns a list of available modes

## 2.54.55 Class Phalcon\Crypt\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Crypt use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.56 Class Phalcon\DI

*implements Phalcon\DiInterface*

Phalcon\DI is a component that implements Dependency Injection/Service Location of services and it's itself a container for them. Since Phalcon is highly decoupled, Phalcon\DI is essential to integrate the different components of the framework. The developer can also use this component to inject dependencies and manage global instances of the different classes used in the application. Basically, this component implements the *Inversion of Control* pattern. Applying this, the objects do not receive their dependencies using setters or constructors, but requesting a service dependency injector. This reduces the overall complexity, since there is only one way to get the required dependencies within a component. Additionally, this pattern increases testability in the code, thus making it less prone to errors.

```
<?php

$di = new Phalcon\DI();

//Using a string definition
$di->set('request', 'Phalcon\Http\Request', true);

//Using an anonymous function
$di->set('request', function(){
    return new Phalcon\Http\Request();
}, true);

$request = $di->getRequest();
```

## Methods

**public \_\_construct ()**

Phalcon\DI constructor

**public *Phalcon\DI\ServiceInterface* set (string \$name, mixed \$definition, [boolean \$shared])**

Registers a service in the services container

**public remove (string \$name)**

Removes a service in the services container

**public mixed getRaw (string \$name)**

Returns a service definition without resolving

**public *Phalcon\DI\ServiceInterface* getService (string \$name)**

Returns a Phalcon\DI\Service instance

**public *Phalcon\DI\ServiceInterface* setService (*Phalcon\DI\ServiceInterface* \$rawDefinition)**

Sets a service using a raw Phalcon\DI\Service definition

**public mixed get (string \$name, [array \$parameters])**

Resolves the service based on its configuration

**public mixed getShared (string \$name, [array \$parameters])**

Resolves a service, the resolved service is stored in the DI, subsequent requests for this service will return the same instance

**public boolean has (string \$name)**

Check whether the DI contains a service by a name

**public boolean wasFreshInstance ()**

Check whether the last service obtained via getShared produced a fresh instance or an existing one

**public *Phalcon\DI\Service* [] getServices ()**

Return the services registered in the DI

**public static setDefault (*Phalcon\DiInterface* \$dependencyInjector)**

Set a default dependency injection container to be obtained into static methods

public static *Phalcon\DiInterface* **getDefault** ()

Return the lastest DI created

public static **reset** ()

Resets the internal default DI

public *Phalcon\DI\ServiceInterface* **attempt** (*string \$name, mixed \$definition, [boolean \$shared]*)

Attempts to register a service in the services container Only is successful if a service hasn't been registered previously with the same name

public *Phalcon\DI\ServiceInterface* **setShared** (*string \$name, mixed \$definition*)

Registers an "always shared" service in the services container

public **setRaw** (*unknown \$rawDefinition*)

...

public *boolean offsetExists* (*unknown \$property*)

Check if a service is registered using the array syntax. Alias for Phalcon\Di::has()

public **offsetSet** (*unknown \$property, unknown \$value*)

Allows to register a shared service using the array syntax. Alias for Phalcon\Di::setShared()

<?php

```
$di[‘request’] = new Phalcon\Http\Request();
```

public *mixed offsetGet* (*unknown \$property*)

Allows to obtain a shared service using the array syntax. Alias for Phalcon\Di::getShared()

<?php

```
var_dump($di[‘request’]);
```

public **offsetUnset** (*unknown \$property*)

Removes a service from the services container using the array syntax. Alias for Phalcon\Di::remove()

public *mixed \_\_call* (*string \$method, [array \$arguments]*)

Magic method to get or set services using setters/getters

public **\_\_clone** ()

...

## 2.54.57 Class Phalcon\DI\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\DI will use this class

### Methods

final private *Exception \_\_clone* () inherited from Exception

Clone the exception

public **`__construct`** (*[string \$message]*, *[int \$code]*, *[Exception \$previous]*) inherited from Exception  
 Exception constructor

final public *string getMessage ()* inherited from Exception  
 Gets the Exception message

final public *int getCode ()* inherited from Exception  
 Gets the Exception code

final public *string getFile ()* inherited from Exception  
 Gets the file in which the exception occurred

final public *int getLine ()* inherited from Exception  
 Gets the line in which the exception occurred

final public *array getTrace ()* inherited from Exception  
 Gets the stack trace

final public *Exception getPrevious ()* inherited from Exception  
 Returns previous Exception

final public *Exception getTraceAsString ()* inherited from Exception  
 Gets the stack trace as a string

public *string \_\_toString ()* inherited from Exception  
 String representation of the exception

### 2.54.58 Class Phalcon\DI\FactoryDefault

*extends class Phalcon\DI*  
*implements Phalcon\DiInterface*

This is a variant of the standard Phalcon\DI. By default it automatically registers all the services provided by the framework. Thanks to this, the developer does not need to register each service individually providing a full stack framework

#### Methods

public **`__construct ()`**  
 Phalcon\DI\FactoryDefault constructor

public *Phalcon\DI\ServiceInterface set (string \$name, mixed \$definition, [boolean \$shared])* inherited from Phalcon\DI  
 Registers a service in the services container

public **`remove (string $name)`** inherited from Phalcon\DI  
 Removes a service in the services container

public *mixed getRaw (string \$name)* inherited from Phalcon\DI  
 Returns a service definition without resolving

public *Phalcon\DI\ServiceInterface getService (string \$name)* inherited from Phalcon\DI

Returns a Phalcon\DI\Service instance

public *Phalcon\DI\ServiceInterface* **setService** (*Phalcon\DI\ServiceInterface* \$rawDefinition) inherited from Phalcon\DI

Sets a service using a raw Phalcon\DI\Service definition

public *mixed* **get** (*string* \$name, [*array* \$parameters]) inherited from Phalcon\DI

Resolves the service based on its configuration

public *mixed* **getShared** (*string* \$name, [*array* \$parameters]) inherited from Phalcon\DI

Resolves a service, the resolved service is stored in the DI, subsequent requests for this service will return the same instance

public *boolean* **has** (*string* \$name) inherited from Phalcon\DI

Check whether the DI contains a service by a name

public *boolean* **wasFreshInstance** () inherited from Phalcon\DI

Check whether the last service obtained via getShared produced a fresh instance or an existing one

public *Phalcon\DI\Service* [] **getServices** () inherited from Phalcon\DI

Return the services registered in the DI

public static **setDefault** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\DI

Set a default dependency injection container to be obtained into static methods

public static *Phalcon\DiInterface* **getDefault** () inherited from Phalcon\DI

Return the lastest DI created

public static **reset** () inherited from Phalcon\DI

Resets the internal default DI

public *Phalcon\DI\ServiceInterface* **attempt** (*string* \$name, *mixed* \$definition, [*boolean* \$shared]) inherited from Phalcon\DI

Attempts to register a service in the services container Only is successful if a service hasn't been registered previously with the same name

public *Phalcon\DI\ServiceInterface* **setShared** (*string* \$name, *mixed* \$definition) inherited from Phalcon\DI

Registers an "always shared" service in the services container

public **setRaw** (*unknown* \$rawDefinition) inherited from Phalcon\DI

...

public *boolean* **offsetExists** (*unknown* \$property) inherited from Phalcon\DI

Check if a service is registered using the array syntax. Alias for Phalcon\Di::has()

public **offsetSet** (*unknown* \$property, *unknown* \$value) inherited from Phalcon\DI

Allows to register a shared service using the array syntax. Alias for Phalcon\Di::setShared()

<?php

```
$di[‘request’] = new Phalcon\Http\Request();
```

public *mixed* **offsetGet** (*unknown* \$property) inherited from Phalcon\DI

Allows to obtain a shared service using the array syntax. Alias for Phalcon\Di::getShared()

```
<?php

var_dump($di['request']);

public offsetUnset (unknown $property) inherited from Phalcon\DI
Removes a service from the services container using the array syntax. Alias for Phalcon\Di::remove()

public mixed __call (string $method, [array $arguments]) inherited from Phalcon\DI
Magic method to get or set services using setters/getters

public __clone () inherited from Phalcon\DI
...

...
```

## 2.54.59 Class Phalcon\DI\FactoryDefault\CLI

*extends class Phalcon\DI\FactoryDefault*

*implements Phalcon\DiInterface*

This is a variant of the standard Phalcon\DI. By default it automatically registers all the services provided by the framework. Thanks to this, the developer does not need to register each service individually. This class is specially suitable for CLI applications

### Methods

public \_\_construct ()

Phalcon\DI\FactoryDefault\CLI constructor

public *Phalcon\DI\ServiceInterface* set (string \$name, mixed \$definition, [boolean \$shared]) inherited from Phalcon\DI

Registers a service in the services container

public remove (string \$name) inherited from Phalcon\DI

Removes a service in the services container

public mixed getRaw (string \$name) inherited from Phalcon\DI

Returns a service definition without resolving

public *Phalcon\DI\ServiceInterface* getService (string \$name) inherited from Phalcon\DI

Returns a Phalcon\DI\Service instance

public *Phalcon\DI\ServiceInterface* setService (*Phalcon\DI\ServiceInterface* \$rawDefinition) inherited from Phalcon\DI

Sets a service using a raw Phalcon\DI\Service definition

public mixed get (string \$name, [array \$parameters]) inherited from Phalcon\DI

Resolves the service based on its configuration

public mixed getShared (string \$name, [array \$parameters]) inherited from Phalcon\DI

Resolves a service, the resolved service is stored in the DI, subsequent requests for this service will return the same instance

public boolean has (string \$name) inherited from Phalcon\DI

Check whether the DI contains a service by a name

`public boolean wasFreshInstance ()` inherited from Phalcon\DI

Check whether the last service obtained via getShared produced a fresh instance or an existing one

`public Phalcon\DI\Service [] getServices ()` inherited from Phalcon\DI

Return the services registered in the DI

`public static setDefault (Phalcon\DiInterface $dependencyInjector)` inherited from Phalcon\DI

Set a default dependency injection container to be obtained into static methods

`public static Phalcon\DiInterface getDefault ()` inherited from Phalcon\DI

Return the lastest DI created

`public static reset ()` inherited from Phalcon\DI

Resets the internal default DI

`public Phalcon\DI\ServiceInterface attempt (string $name, mixed $definition, [boolean $shared])` inherited from Phalcon\DI

Attempts to register a service in the services container Only is successful if a service hasn't been registered previously with the same name

`public Phalcon\DI\ServiceInterface setShared (string $name, mixed $definition)` inherited from Phalcon\DI

Registers an "always shared" service in the services container

`public setRaw (unknown $rawDefinition)` inherited from Phalcon\DI

...

`public boolean offsetExists (unknown $property)` inherited from Phalcon\DI

Check if a service is registered using the array syntax. Alias for Phalcon\Di::has()

`public offsetSet (unknown $property, unknown $value)` inherited from Phalcon\DI

Allows to register a shared service using the array syntax. Alias for Phalcon\Di::setShared()

`<?php`

```
$di[‘request’] = new Phalcon\Http\Request();
```

`public mixed offsetGet (unknown $property)` inherited from Phalcon\DI

Allows to obtain a shared service using the array syntax. Alias for Phalcon\Di::getShared()

`<?php`

```
var_dump($di[‘request’]);
```

`public offsetUnset (unknown $property)` inherited from Phalcon\DI

Removes a service from the services container using the array syntax. Alias for Phalcon\Di::remove()

`public mixed __call (string $method, [array $arguments])` inherited from Phalcon\DI

Magic method to get or set services using setters/getters

`public __clone ()` inherited from Phalcon\DI

...

## 2.54.60 Abstract class Phalcon\DI\Injectable

*implements Phalcon\DI\InjectionAwareInterface, Phalcon\Events\EventsAwareInterface*

This class allows to access services in the services container by just only accessing a public property with the same name of a registered service

### Methods

**public setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

**public Phalcon\DiInterface getDI** ()

Returns the internal dependency injector

**public setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager)

Sets the event manager

**public Phalcon\Events\ManagerInterface getEventsManager** ()

Returns the internal event manager

**public \_\_get** (*unknown* \$property)

Magic method \_\_get

## 2.54.61 Class Phalcon\DI\Service

*implements Phalcon\DI\ServiceInterface*

Represents individually a service in the services container

<?php

```
$service = new Phalcon\DI\Service('request', 'Phalcon\Http\Request');
$request = $service->resolve();
```

<?php

### Methods

**public \_\_construct** (*string* \$name, *mixed* \$definition, [*boolean* \$shared])

**public getName** ()

Returns the service's name

**public setShared** (*boolean* \$shared)

Sets if the service is shared or not

**public boolean isShared** ()

Check whether the service is shared or not

**public setSharedInstance** (*mixed* \$sharedInstance)

Sets/Resets the shared instance related to the service

public **setDefinition** (*mixed* \$definition)

Set the service definition

public *mixed* **getDefinition** ()

Returns the service definition

public *object* **resolve** ([*array* \$parameters], [*Phalcon\DiInterface* \$dependencyInjector])

Resolves the service

public *Phalcon\DI\Service* **setParameter** (*long* \$position, *array* \$parameter)

Changes a parameter in the definition without resolve the service

public *array* **getParameter** (*int* \$position)

Returns a parameter in a specific position

public *bool* **isResolved** ()

Returns true if the service was resolved

public static *Phalcon\DI\Service* **\_\_set\_state** ([*unknown* \$properties])

Restore the internal state of a service

## 2.54.62 Class Phalcon\DI\Service\Builder

This class builds instances based on complex definitions

### Methods

protected *mixed* **\_buildParameter** ()

Resolves a constructor/call parameter

protected *array* **\_buildParameters** ()

Resolves an array of parameters

public *mixed* **build** (*Phalcon\DiInterface* \$dependencyInjector, *array* \$definition, [*array* \$parameters])

Builds a service using a complex service definition

## 2.54.63 Abstract class Phalcon\Db

Phalcon\Db and its related classes provide a simple SQL database interface for Phalcon Framework. The Phalcon\Db is the basic class you use to connect your PHP application to an RDBMS. There is a different adapter class for each brand of RDBMS. This component is intended to lower level database operations. If you want to interact with databases using higher level of abstraction use Phalcon\Mvc\Model. Phalcon\Db is an abstract class. You only can use it with a database adapter like Phalcon\Db\Adapter\Pdo

```
<?php
```

```
try {
```

```
$connection = new Phalcon\Db\Adapter\Pdo\Mysql(array(
    'host' => '192.168.0.11',
    'username' => 'sigma',
```

```

'password' => 'secret',
'dbname' => 'blog',
'port' => '3306',
));

$result = $connection->query("SELECT * FROM robots LIMIT 5");
$result->setFetchMode(Phalcon\Db::FETCH_NUM);
while ($robot = $result->fetch()) {
    print_r($robot);
}

} catch (Phalcon\Db\Exception $e) {
echo $e->getMessage(), PHP_EOL;
}

```

## Constants

*integer* **FETCH\_USE\_DEFAULT**  
*integer* **FETCH\_LAZY**  
*integer* **FETCH\_ASSOC**  
*integer* **FETCH\_NUM**  
*integer* **FETCH\_BOTH**  
*integer* **FETCH\_OBJ**  
*integer* **FETCH\_BOUND**  
*integer* **FETCH\_COLUMN**  
*integer* **FETCH\_CLASS**  
*integer* **FETCH\_INTO**  
*integer* **FETCH\_FUNC**  
*integer* **FETCH\_NAMED**  
*integer* **FETCH\_KEY\_PAIR**  
*integer* **FETCH\_GROUP**  
*integer* **FETCH\_UNIQUE**  
*integer* **FETCH\_CLASSTYPE**  
*integer* **FETCH\_SERIALIZE**  
*integer* **FETCH\_PROPS\_LATE**

## Methods

**public static `setup` (*array* `$options`)**

Enables/disables options in the Database component

## 2.54.64 Abstract class Phalcon\Db\Adapter

implements *Phalcon\Events\EventsAwareInterface*, *Phalcon\Db\AdapterInterface*

Base class for Phalcon\Db adapters

### Methods

protected **\_\_construct** ()

Phalcon\Db\Adapter constructor

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager)

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** ()

Returns the internal event manager

public **setDialect** (*unknown* \$dialect)

Sets the dialect used to produce the SQL

public *Phalcon\Db\DialectInterface* **getDialect** ()

Returns internal dialect instance

public *array* **fetchOne** (*string* \$sqlQuery, [*int* \$fetchMode], [*unknown* \$placeholders])

Returns the first row in a SQL query result

<?php

```
//Getting first robot
$robot = $connection->fetchOne("SELECT * FROM robots");
print_r($robot);
```

```
//Getting first robot with associative indexes only
```

```
$robot = $connection->fetchOne("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
print_r($robot);
```

public *array* **fetchAll** (*string* \$sqlQuery, [*int* \$fetchMode], [*unknown* \$placeholders])

Dumps the complete result of a query into an array

<?php

```
//Getting all robots with associative indexes only
$robots = $connection->fetchAll("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
foreach ($robots as $robot) {
    print_r($robot);
}
```

```
//Getting all robots that contains word "robot" withing the name
```

```
$robots = $connection->fetchAll("SELECT * FROM robots WHERE name LIKE :name",
    Phalcon\Db::FETCH_ASSOC,
    array('name' => '%robot%')
);
foreach($robots as $robot){
    print_r($robot);
}
```

`public boolean insert (string $table, array $values, [array $fields], [array $dataTypes])`

Inserts data into a table using custom RBDM SQL syntax

`<?php`

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Next SQL sentence is sent to the database system
INSERT INTO `robots` ('name', 'year') VALUES ("Astro boy", 1952);
```

`public boolean update (string $table, array $fields, array $values, [string $whereCondition], [array $dataTypes])`

Updates data on a table using custom RBDM SQL syntax

`<?php`

```
//Updating existing robot
$success = $connection->update(
    "robots",
    array("name"),
    array("New Astro Boy"),
    "id = 101"
);

//Next SQL sentence is sent to the database system
UPDATE `robots` SET `name` = "Astro boy" WHERE id = 101
```

`public boolean delete (string $table, [string $whereCondition], [array $placeholders], [array $dataTypes])`

Deletes data from a table using custom RBDM SQL syntax

`<?php`

```
//Deleting existing robot
$success = $connection->delete(
    "robots",
    "id = 101"
);

//Next SQL sentence is generated
DELETE FROM `robots` WHERE `id` = 101
```

`public string getColumnList (array $columnList)`

Gets a list of columns

`public string limit (string $sqlQuery, int $number)`

Appends a LIMIT clause to \$sqlQuery argument

`<?php`

```
echo $connection->limit("SELECT * FROM robots", 5);
```

`public string tableExists (string $tableName, [string $schemaName])`

Generates SQL checking for the existence of a schema.table

```
<?php
```

```
var_dump($connection->tableExists("blog", "posts"));
```

public *string* **viewExists** (*string* \$viewName, [*string* \$schemaName])

Generates SQL checking for the existence of a schema.view

```
<?php
```

```
var_dump($connection->viewExists("active_users", "posts"));
```

public *string* **forUpdate** (*string* \$sqlQuery)

Returns a SQL modified with a FOR UPDATE clause

public *string* **sharedLock** (*string* \$sqlQuery)

Returns a SQL modified with a LOCK IN SHARE MODE clause

public *boolean* **createTable** (*string* \$tableName, *string* \$schemaName, *array* \$definition)

Creates a table

public *boolean* **dropTable** (*string* \$tableName, [*string* \$schemaName], [*boolean* \$IfExists])

Drops a table from a schema/database

public *boolean* **createView** (*unknown* \$viewName, *array* \$definition, [*string* \$schemaName])

Creates a view

public *boolean* **dropView** (*string* \$viewName, [*string* \$schemaName], [*boolean* \$IfExists])

Drops a view

public *boolean* **addColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column)

Adds a column to a table

public *boolean* **modifyColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column)

Modifies a table column based on a definition

public *boolean* **dropColumn** (*string* \$tableName, *string* \$schemaName, *string* \$columnName)

Drops a column from a table

public *boolean* **addIndex** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\IndexInterface* \$index)

Adds an index to a table

public *boolean* **dropIndex** (*string* \$tableName, *string* \$schemaName, *string* \$indexName)

Drop an index from a table

public *boolean* **addPrimaryKey** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\IndexInterface* \$index)

Adds a primary key to a table

public *boolean* **dropPrimaryKey** (*string* \$tableName, *string* \$schemaName)

Drops a table's primary key

```

public boolean true addForeignKey (string $tableName, string $schemaName,
Phalcon\Db\ReferenceInterface $reference)

Adds a foreign key to a table

public boolean true dropForeignKey (string $tableName, string $schemaName, string $referenceName)

Drops a foreign key from a table

public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)

Returns the SQL column definition from a column

public array listTables ([string $schemaName])

List all tables on a database

<?php

print_r($connection->listTables("blog"));

public array listViews ([string $schemaName])

List all views on a database

<?php

print_r($connection->listViews("blog")); ?>

public Phalcon\Db\Index [] describeIndexes (string $table, [string $schema])

Lists table indexes

<?php

print_r($connection->describeIndexes('robots_parts'));

public Phalcon\Db\Reference [] describeReferences (string $table, [string $schema])

Lists table references

<?php

print_r($connection->describeReferences('robots_parts'));

public array tableOptions (string $tableName, [string $schemaName])

Gets creation options from a table

<?php

print_r($connection->tableOptions('robots'));

public boolean createSavepoint (string $name)

Creates a new savepoint

public boolean releaseSavepoint (string $name)

Releases given savepoint

public boolean rollbackSavepoint (string $name)

Rollbacks given savepoint

```

```
public      Phalcon\Db\AdapterInterface    setNestedTransactionsWithSavepoints  (boolean
$nestedTransactionsWithSavepoints)

Set if nested transactions should use savepoints

public boolean isNestedTransactionsWithSavepoints ()

Returns if nested transactions should use savepoints

public string getNestedTransactionSavepointName ()

Returns the savepoint name to use for nested transactions

public Phalcon\Db\RawValue getDefaultIdValue ()

Returns the default identity value to be inserted in an identity column

<?php

//Inserting a new robot with a valid default value for the column 'id'
$success = $connection->insert(
    "robots",
    array($connection->getDefaultIdValue(), "Astro Boy", 1952),
    array("id", "name", "year")
);

public boolean supportSequences ()

Check whether the database system requires a sequence to produce auto-numeric values

public boolean useExplicitIdValue ()

Check whether the database system requires an explicit value for identity columns

public array getDescriptor ()

Return descriptor used to connect to the active database

public string getConnectionId ()

Gets the active connection unique identifier

public string getSQLStatement ()

Active SQL statement in the object

public string getRealSQLStatement ()

Active SQL statement in the object without replace bound paramters

public array getSQLVariables ()

Active SQL statement in the object

public array getSQLBindTypes ()

Active SQL statement in the object

public string getType ()

Returns type of database system the adapter is used for

public string getDialectType ()

Returns the name of the dialect used

abstract public boolean connect ([array $descriptor]) inherited from Phalcon\Db\AdapterInterface
```

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection

`abstract public Phalcon\Db\ResultInterface query (string $sqlStatement, [array $placeholders], [array $dataTypes])` inherited from Phalcon\Db\AdapterInterface

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server return rows

`abstract public boolean execute (string $sqlStatement, [array $placeholders], [array $dataTypes])` inherited from Phalcon\Db\AdapterInterface

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server don't return any row

`abstract public int affectedRows ()` inherited from Phalcon\Db\AdapterInterface

Returns the number of affected rows by the last INSERT/UPDATE/DELETE reported by the database system

`abstract public boolean close ()` inherited from Phalcon\Db\AdapterInterface

Closes active connection returning success. Phalcon automatically closes and destroys active connections within Phalcon\Db\Pool

`abstract public string escapeIdentifier (string $identifier)` inherited from Phalcon\Db\AdapterInterface

Escapes a column/table/schema name

`abstract public string escapeString (string $str)` inherited from Phalcon\Db\AdapterInterface

Escapes a value to avoid SQL injections

`abstract public array convertBoundParams (string $sqlStatement, array $params)` inherited from Phalcon\Db\AdapterInterface

Converts bound params like :name: or ?1 into ? bind params

`abstract public int lastInsertId ([string $sequenceName])` inherited from Phalcon\Db\AdapterInterface

Returns insert id for the auto\_increment column inserted in the last SQL statement

`abstract public boolean begin ()` inherited from Phalcon\Db\AdapterInterface

Starts a transaction in the connection

`abstract public boolean rollback ()` inherited from Phalcon\Db\AdapterInterface

Rollbacks the active transaction in the connection

`abstract public boolean commit ()` inherited from Phalcon\Db\AdapterInterface

Commits the active transaction in the connection

`abstract public boolean isUnderTransaction ()` inherited from Phalcon\Db\AdapterInterface

Checks whether connection is under database transaction

`abstract public PDO getInternalHandler ()` inherited from Phalcon\Db\AdapterInterface

Return internal PDO handler

`abstract public Phalcon\Db\ColumnInterface [] describeColumns (string $table, [string $schema])` inherited from Phalcon\Db\AdapterInterface

Returns an array of Phalcon\Db\Column objects describing a table

## 2.54.65 Abstract class Phalcon\Db\Adapter\Pdo

*extends abstract class Phalcon\Db\Adapter*

*implements Phalcon\Db\AdapterInterface, Phalcon\Events\EventsAwareInterface*

Phalcon\Db\Adapter\Pdo is the Phalcon\Db that internally uses PDO to connect to a database

<?php

```
$connection = new Phalcon\Db\Adapter\Pdo\Mysql(array(
    'host' => '192.168.0.11',
    'username' => 'sigma',
    'password' => 'secret',
    'dbname' => 'blog',
    'port' => '3306'
));
```

### Methods

public **\_\_construct** (*array \$descriptor*)

Constructor for Phalcon\Db\Adapter\Pdo

public **boolean connect** (*[array \$descriptor]*)

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection

<?php

```
//Make a connection
$connection = new Phalcon\Db\Adapter\Pdo\Mysql(array(
    'host' => '192.168.0.11',
    'username' => 'sigma',
    'password' => 'secret',
    'dbname' => 'blog',
));
//Reconnect
$connection->connect();
```

public **PDOStatement prepare** (*string \$sqlStatement*)

Returns a PDO prepared statement to be executed with 'executePrepared'

<?php

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

public **PDOStatement executePrepared** (*PDOStatement \$statement, array \$placeholders, array \$dataTypes*)

Executes a prepared statement binding. This function uses integer indexes starting from zero

<?php

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

`public Phalcon\Db\ResultInterface query (string $sqlStatement, [unknown $placeholders], [unknown $dataTypes])`

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server is returning rows

```
<?php
```

```
//Querying data
$resultset = $connection->query("SELECT * FROM robots WHERE type='mechanical'");
$resultset = $connection->query("SELECT * FROM robots WHERE type=?", array("mechanical"));
```

`public boolean execute (string $sqlStatement, [unknown $placeholders], [unknown $dataTypes])`

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server doesn't return any row

```
<?php
```

```
//Inserting data
$success = $connection->execute("INSERT INTO robots VALUES (1, 'Astro Boy')");
$success = $connection->execute("INSERT INTO robots VALUES (?, ?)", array(1, 'Astro Boy'));
```

`public int affectedRows ()`

Returns the number of affected rows by the lastest INSERT/UPDATE/DELETE executed in the database system

```
<?php
```

```
$connection->execute("DELETE FROM robots");
echo $connection->affectedRows(), ' were deleted';
```

`public boolean close ()`

Closes the active connection returning success. Phalcon automatically closes and destroys active connections when the request ends

`public string escapeIdentifier (string $identifier)`

Escapes a column/table/schema name

```
<?php
```

```
$escapedTable = $connection->escapeIdentifier('robots');
$escapedTable = $connection->escapeIdentifier(array('store', 'robots'));
```

`public string escapeString (string $str)`

Escapes a value to avoid SQL injections according to the active charset in the connection

```
<?php
```

```
$escapedStr = $connection->escapeString('some dangerous value');
```

`public array convertBoundParams (unknown $sqlStatement, array $params)`

Converts bound parameters such as :name: or ?1 into PDO bind params ?

```
<?php
```

```
print_r($connection->convertBoundParams('SELECT * FROM robots WHERE name = :name:', array('Bender')));
```

public *int* **lastInsertId** ([*string* \$sequenceName])

Returns the insert id for the auto\_increment/serial column inserted in the lastest executed SQL statement

<?php

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);
```

```
//Getting the generated id
$id = $connection->lastInsertId();
```

public *boolean* **begin** ([*boolean* \$nesting])

Starts a transaction in the connection

public *boolean* **rollback** ([*boolean* \$nesting])

Rollbacks the active transaction in the connection

public *boolean* **commit** ([*boolean* \$nesting])

Commits the active transaction in the connection

public *int* **getTransactionLevel** ()

Returns the current transaction nesting level

public *boolean* **isUnderTransaction** ()

Checks whether the connection is under a transaction

<?php

```
$connection->begin();
var_dump($connection->isUnderTransaction()); //true
```

public *PDO* **getInternalHandler** ()

Return internal PDO handler

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\Db\Adapter

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\Db\Adapter

Returns the internal event manager

public **setDialect** (*unknown* \$dialect) inherited from Phalcon\Db\Adapter

Sets the dialect used to produce the SQL

public *Phalcon\Db\DialectInterface* **getDialect** () inherited from Phalcon\Db\Adapter

Returns internal dialect instance

public *array* **fetchOne** (*string* \$sqlQuery, [*int* \$fetchMode], [*unknown* \$placeholders]) inherited from Phalcon\Db\Adapter

Returns the first row in a SQL query result

```
<?php
```

```
//Getting first robot
$robot = $connection->fetchOne("SELECT * FROM robots");
print_r($robot);

//Getting first robot with associative indexes only
$robot = $connection->fetchOne("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
print_r($robot);
```

**public array fetchAll (string \$sqlQuery, [int \$fetchMode], [unknown \$placeholders])** inherited from Phalcon\Db\Adapter

Dumps the complete result of a query into an array

```
<?php
```

```
//Getting all robots with associative indexes only
$robots = $connection->fetchAll("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
foreach ($robots as $robot) {
    print_r($robot);
}

//Getting all robots that contains word "robot" withing the name
$robots = $connection->fetchAll("SELECT * FROM robots WHERE name LIKE :name",
    Phalcon\Db::FETCH_ASSOC,
    array('name' => '%robot%')
);
foreach($robots as $robot){
    print_r($robot);
}
```

**public boolean insert (string \$table, array \$values, [array \$fields], [array \$dataTypes])** inherited from Phalcon\Db\Adapter

Inserts data into a table using custom RBDM SQL syntax

```
<?php
```

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Next SQL sentence is sent to the database system
INSERT INTO `robots` ('name', 'year') VALUES ("Astro boy", 1952);
```

**public boolean update (string \$table, array \$fields, array \$values, [string \$whereCondition], [array \$dataTypes])** inherited from Phalcon\Db\Adapter

Updates data on a table using custom RBDM SQL syntax

```
<?php
```

```
//Updating existing robot
$success = $connection->update(
    "robots",
    array("name"),
```

```
array("New Astro Boy"),
      "id = 101"
);

//Next SQL sentence is sent to the database system
UPDATE `robots` SET `name` = "Astro boy" WHERE id = 101
```

public *boolean delete* (*string \$table*, [*string \$whereCondition*], [*array \$placeholders*], [*array \$dataTypes*])  
inherited from Phalcon\Db\Adapter

Deletes data from a table using custom RBDM SQL syntax

```
<?php

//Deleting existing robot
$success = $connection->delete(
    "robots",
    "id = 101"
);

//Next SQL sentence is generated
DELETE FROM `robots` WHERE `id` = 101
```

public *string getColumnList* (*array \$columnList*) inherited from Phalcon\Db\Adapter

Gets a list of columns

public *string limit* (*string \$sqlQuery*, *int \$number*) inherited from Phalcon\Db\Adapter

Appends a LIMIT clause to \$sqlQuery argument

```
<?php

echo $connection->limit("SELECT * FROM robots", 5);
```

public *string tableExists* (*string \$tableName*, [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.table

```
<?php

var_dump($connection->tableExists("blog", "posts"));
```

public *string viewExists* (*string \$viewName*, [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.view

```
<?php

var_dump($connection->viewExists("active_users", "posts"));
```

public *string forUpdate* (*string \$sqlQuery*) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a FOR UPDATE clause

public *string sharedLock* (*string \$sqlQuery*) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a LOCK IN SHARE MODE clause

public *boolean createTable* (*string \$tableName*, *string \$schemaName*, *array \$definition*) inherited from Phalcon\Db\Adapter

Creates a table

`public boolean dropTable (string $tableName, [string $schemaName], [boolean $IfExists])` inherited from Phalcon\Db\Adapter

Drops a table from a schema/database

`public boolean createView (unknown $viewName, array $definition, [string $schemaName])` inherited from Phalcon\Db\Adapter

Creates a view

`public boolean dropView (string $viewName, [string $schemaName], [boolean $IfExists])` inherited from Phalcon\Db\Adapter

Drops a view

`public boolean addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Adds a column to a table

`public boolean modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Modifies a table column based on a definition

`public boolean dropColumn (string $tableName, string $schemaName, string $columnName)` inherited from Phalcon\Db\Adapter

Drops a column from a table

`public boolean addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds an index to a table

`public boolean dropIndex (string $tableName, string $schemaName, string $indexName)` inherited from Phalcon\Db\Adapter

Drop an index from a table

`public boolean addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds a primary key to a table

`public boolean dropPrimaryKey (string $tableName, string $schemaName)` inherited from Phalcon\Db\Adapter

Drops a table's primary key

`public boolean true addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)` inherited from Phalcon\Db\Adapter

Adds a foreign key to a table

`public boolean dropForeignKey (string $tableName, string $schemaName, string $referenceName)` inherited from Phalcon\Db\Adapter

Drops a foreign key from a table

`public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Returns the SQL column definition from a column

`public array listTables ([string $schemaName])` inherited from Phalcon\Db\Adapter

List all tables on a database

```
<?php
```

```
print_r($connection->listTables("blog"));
```

public *array* **listViews** (*[string \$schemaName]*) inherited from Phalcon\Db\Adapter

List all views on a database

```
<?php
```

```
print_r($connection->listViews("blog")); ?>
```

public *Phalcon\Db\Index* [] **describeIndexes** (*string \$table, [string \$schema]*) inherited from Phalcon\Db\Adapter

Lists table indexes

```
<?php
```

```
print_r($connection->describeIndexes('robots_parts'));
```

public *Phalcon\Db\Reference* [] **describeReferences** (*string \$table, [string \$schema]*) inherited from Phalcon\Db\Adapter

Lists table references

```
<?php
```

```
print_r($connection->describeReferences('robots_parts'));
```

public *array* **tableOptions** (*string \$tableName, [string \$schemaName]*) inherited from Phalcon\Db\Adapter

Gets creation options from a table

```
<?php
```

```
print_r($connection->tableOptions('robots'));
```

public *boolean* **createSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Creates a new savepoint

public *boolean* **releaseSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Releases given savepoint

public *boolean* **rollbackSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Rollbacks given savepoint

public *Phalcon\Db\AdapterInterface* **setNestedTransactionsWithSavepoints** (*boolean \$nestedTransactionsWithSavepoints*) inherited from Phalcon\Db\Adapter

Set if nested transactions should use savepoints

public *boolean* **isNestedTransactionsWithSavepoints** () inherited from Phalcon\Db\Adapter

Returns if nested transactions should use savepoints

public *string* **getNestedTransactionSavepointName** () inherited from Phalcon\Db\Adapter

Returns the savepoint name to use for nested transactions

public *Phalcon\Db\RawValue* **getDefaultIdValue** () inherited from Phalcon\Db\Adapter

Returns the default identity value to be inserted in an identity column

```
<?php

//Inserting a new robot with a valid default value for the column 'id'
$success = $connection->insert(
    "robots",
    array($connection->getDefaultIdValue(), "Astro Boy", 1952),
    array("id", "name", "year")
);
```

**public boolean supportSequences ()** inherited from Phalcon\Db\Adapter

Check whether the database system requires a sequence to produce auto-numeric values

**public boolean useExplicitIdValue ()** inherited from Phalcon\Db\Adapter

Check whether the database system requires an explicit value for identity columns

**public array getDescriptor ()** inherited from Phalcon\Db\Adapter

Return descriptor used to connect to the active database

**public string getConnectionId ()** inherited from Phalcon\Db\Adapter

Gets the active connection unique identifier

**public string getSQLStatement ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object

**public string getRealSQLStatement ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object without replace bound paramters

**public array getSQLVariables ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object

**public array getSQLBindTypes ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object

**public string getType ()** inherited from Phalcon\Db\Adapter

Returns type of database system the adapter is used for

**public string getDialectType ()** inherited from Phalcon\Db\Adapter

Returns the name of the dialect used

**abstract public Phalcon\Db\ColumnInterface [] describeColumns (string \$table, [string \$schema])**  
inherited from Phalcon\Db\AdapterInterface

Returns an array of Phalcon\Db\Column objects describing a table

## 2.54.66 Class Phalcon\Db\Adapter\Pdo\Mysql

*extends abstract class Phalcon\Db\Adapter\Pdo*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\Db\AdapterInterface*

Specific functions for the Mysql database system

```
<?php

$config = array(
    "host" => "192.168.0.11",
    "dbname" => "blog",
    "port" => 3306,
    "username" => "sigma",
    "password" => "secret"
);

$connection = new Phalcon\Db\Adapter\Pdo\Mysql($config);
```

## Methods

public *string* escapeIdentifier (*string* \$identifier)

Escapes a column/table/schema name

public *Phalcon\Db\Column* [] describeColumns (*string* \$table, [*string* \$schema])

Returns an array of Phalcon\Db\Column objects describing a table

```
<?php
```

```
print_r($connection->describeColumns("posts")); ?>
```

public \_\_construct (*array* \$descriptor) inherited from Phalcon\Db\Adapter\Pdo

Constructor for Phalcon\Db\Adapter\Pdo

public *boolean* connect ([*array* \$descriptor]) inherited from Phalcon\Db\Adapter\Pdo

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection

```
<?php
```

```
//Make a connection
$connection = new Phalcon\Db\Adapter\Pdo\Mysql(array(
    'host' => '192.168.0.11',
    'username' => 'sigma',
    'password' => 'secret',
    'dbname' => 'blog',
));
```

```
//Reconnect
```

```
$connection->connect();
```

public *PDOStatement* prepare (*string* \$sqlStatement) inherited from Phalcon\Db\Adapter\Pdo

Returns a PDO prepared statement to be executed with 'executePrepared'

```
<?php
```

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

public *PDOStatement* executePrepared (*PDOStatement* \$statement, *array* \$placeholders, *array* \$dataTypes) inherited from Phalcon\Db\Adapter\Pdo

Executes a prepared statement binding. This function uses integer indexes starting from zero

```
<?php
```

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

*public Phalcon\Db\ResultInterface query (string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes])* inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server is returning rows

```
<?php
```

```
//Querying data
$resultset = $connection->query("SELECT * FROM robots WHERE type='mechanical'");
$resultset = $connection->query("SELECT * FROM robots WHERE type=?", array("mechanical"));
```

*public boolean execute (string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes])* inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server doesn't return any row

```
<?php
```

```
//Inserting data
$success = $connection->execute("INSERT INTO robots VALUES (1, 'Astro Boy')");
$success = $connection->execute("INSERT INTO robots VALUES (?, ?)", array(1, 'Astro Boy'));
```

*public int affectedRows ()* inherited from Phalcon\Db\Adapter\Pdo

Returns the number of affected rows by the lastest INSERT/UPDATE/DELETE executed in the database system

```
<?php
```

```
$connection->execute("DELETE FROM robots");
echo $connection->affectedRows(), ' were deleted';
```

*public boolean close ()* inherited from Phalcon\Db\Adapter\Pdo

Closes the active connection returning success. Phalcon automatically closes and destroys active connections when the request ends

*public string escapeString (string \$str)* inherited from Phalcon\Db\Adapter\Pdo

Escapes a value to avoid SQL injections according to the active charset in the connection

```
<?php
```

```
$escapedStr = $connection->escapeString('some dangerous value');
```

*public array convertBoundParams (unknown \$sqlStatement, array \$params)* inherited from Phalcon\Db\Adapter\Pdo

Converts bound parameters such as :name: or ?1 into PDO bind params ?

```
<?php
```

```
print_r($connection->convertBoundParams('SELECT * FROM robots WHERE name = :name:', array('Bender')));
```

public *int* **lastInsertId** ([*string* \$sequenceName]) inherited from Phalcon\Db\Adapter\Pdo

Returns the insert id for the auto\_increment/serial column inserted in the lastest executed SQL statement

<?php

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Getting the generated id
$id = $connection->lastInsertId();
```

public *boolean* **begin** ([*boolean* \$nesting]) inherited from Phalcon\Db\Adapter\Pdo

Starts a transaction in the connection

public *boolean* **rollback** ([*boolean* \$nesting]) inherited from Phalcon\Db\Adapter\Pdo

Rollbacks the active transaction in the connection

public *boolean* **commit** ([*boolean* \$nesting]) inherited from Phalcon\Db\Adapter\Pdo

Commits the active transaction in the connection

public *int* **getTransactionLevel** () inherited from Phalcon\Db\Adapter\Pdo

Returns the current transaction nesting level

public *boolean* **isUnderTransaction** () inherited from Phalcon\Db\Adapter\Pdo

Checks whether the connection is under a transaction

<?php

```
$connection->begin();
var_dump($connection->isUnderTransaction()); //true
```

public *PDO* **getInternalHandler** () inherited from Phalcon\Db\Adapter\Pdo

Return internal PDO handler

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\Db\Adapter

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\Db\Adapter

Returns the internal event manager

public **setDialect** (*unknown* \$dialect) inherited from Phalcon\Db\Adapter

Sets the dialect used to produce the SQL

public *Phalcon\Db\ManagerInterface* **getDialect** () inherited from Phalcon\Db\Adapter

Returns internal dialect instance

public *array* **fetchOne** (*string* \$sqlQuery, [*int* \$fetchMode], [*unknown* \$placeholders]) inherited from Phalcon\Db\Adapter

Returns the first row in a SQL query result

```
<?php
```

```
//Getting first robot
$robot = $connection->fetchOne("SELECT * FROM robots");
print_r($robot);

//Getting first robot with associative indexes only
$robot = $connection->fetchOne("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
print_r($robot);
```

**public array fetchAll (string \$sqlQuery, [int \$fetchMode], [unknown \$placeholders])** inherited from Phalcon\Db\Adapter

Dumps the complete result of a query into an array

```
<?php
```

```
//Getting all robots with associative indexes only
$robots = $connection->fetchAll("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
foreach ($robots as $robot) {
    print_r($robot);
}

//Getting all robots that contains word "robot" withing the name
$robots = $connection->fetchAll("SELECT * FROM robots WHERE name LIKE :name",
    Phalcon\Db::FETCH_ASSOC,
    array('name' => '%robot%')
);
foreach($robots as $robot){
    print_r($robot);
}
```

**public boolean insert (string \$table, array \$values, [array \$fields], [array \$dataTypes])** inherited from Phalcon\Db\Adapter

Inserts data into a table using custom RBDM SQL syntax

```
<?php
```

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Next SQL sentence is sent to the database system
INSERT INTO `robots` ('name', 'year') VALUES ("Astro boy", 1952);
```

**public boolean update (string \$table, array \$fields, array \$values, [string \$whereCondition], [array \$dataTypes])** inherited from Phalcon\Db\Adapter

Updates data on a table using custom RBDM SQL syntax

```
<?php
```

```
//Updating existing robot
$success = $connection->update(
    "robots",
    array("name"),
```

```
array("New Astro Boy"),
      "id = 101"
);

//Next SQL sentence is sent to the database system
UPDATE `robots` SET `name` = "Astro boy" WHERE id = 101
```

public *boolean delete* (*string \$table*, [*string \$whereCondition*], [*array \$placeholders*], [*array \$dataTypes*])  
inherited from Phalcon\Db\Adapter

Deletes data from a table using custom RBDM SQL syntax

```
<?php
```

```
//Deleting existing robot
$success = $connection->delete(
    "robots",
    "id = 101"
);

//Next SQL sentence is generated
DELETE FROM `robots` WHERE `id` = 101
```

public *string getColumnList* (*array \$columnList*) inherited from Phalcon\Db\Adapter

Gets a list of columns

public *string limit* (*string \$sqlQuery*, *int \$number*) inherited from Phalcon\Db\Adapter

Appends a LIMIT clause to \$sqlQuery argument

```
<?php
```

```
echo $connection->limit("SELECT * FROM robots", 5);
```

public *string tableExists* (*string \$tableName*, [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.table

```
<?php
```

```
var_dump($connection->tableExists("blog", "posts"));
```

public *string viewExists* (*string \$viewName*, [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.view

```
<?php
```

```
var_dump($connection->viewExists("active_users", "posts"));
```

public *string forUpdate* (*string \$sqlQuery*) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a FOR UPDATE clause

public *string sharedLock* (*string \$sqlQuery*) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a LOCK IN SHARE MODE clause

public *boolean createTable* (*string \$tableName*, *string \$schemaName*, *array \$definition*) inherited from Phalcon\Db\Adapter

Creates a table

`public boolean dropTable (string $tableName, [string $schemaName], [boolean $IfExists])` inherited from Phalcon\Db\Adapter

Drops a table from a schema/database

`public boolean createView (unknown $viewName, array $definition, [string $schemaName])` inherited from Phalcon\Db\Adapter

Creates a view

`public boolean dropView (string $viewName, [string $schemaName], [boolean $IfExists])` inherited from Phalcon\Db\Adapter

Drops a view

`public boolean addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Adds a column to a table

`public boolean modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Modifies a table column based on a definition

`public boolean dropColumn (string $tableName, string $schemaName, string $columnName)` inherited from Phalcon\Db\Adapter

Drops a column from a table

`public boolean addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds an index to a table

`public boolean dropIndex (string $tableName, string $schemaName, string $indexName)` inherited from Phalcon\Db\Adapter

Drop an index from a table

`public boolean addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds a primary key to a table

`public boolean dropPrimaryKey (string $tableName, string $schemaName)` inherited from Phalcon\Db\Adapter

Drops a table's primary key

`public boolean true addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)` inherited from Phalcon\Db\Adapter

Adds a foreign key to a table

`public boolean dropForeignKey (string $tableName, string $schemaName, string $referenceName)` inherited from Phalcon\Db\Adapter

Drops a foreign key from a table

`public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Returns the SQL column definition from a column

`public array listTables ([string $schemaName])` inherited from Phalcon\Db\Adapter

List all tables on a database

```
<?php
```

```
print_r($connection->listTables("blog"));
```

public *array* **listViews** (*[string \$schemaName]*) inherited from Phalcon\Db\Adapter

List all views on a database

```
<?php
```

```
print_r($connection->listViews("blog")); ?>
```

public *Phalcon\Db\Index* [] **describeIndexes** (*string \$table, [string \$schema]*) inherited from Phalcon\Db\Adapter

Lists table indexes

```
<?php
```

```
print_r($connection->describeIndexes('robots_parts'));
```

public *Phalcon\Db\Reference* [] **describeReferences** (*string \$table, [string \$schema]*) inherited from Phalcon\Db\Adapter

Lists table references

```
<?php
```

```
print_r($connection->describeReferences('robots_parts'));
```

public *array* **tableOptions** (*string \$tableName, [string \$schemaName]*) inherited from Phalcon\Db\Adapter

Gets creation options from a table

```
<?php
```

```
print_r($connection->tableOptions('robots'));
```

public *boolean* **createSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Creates a new savepoint

public *boolean* **releaseSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Releases given savepoint

public *boolean* **rollbackSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Rollbacks given savepoint

public *Phalcon\Db\AdapterInterface* **setNestedTransactionsWithSavepoints** (*boolean \$nestedTransactionsWithSavepoints*) inherited from Phalcon\Db\Adapter

Set if nested transactions should use savepoints

public *boolean* **isNestedTransactionsWithSavepoints** () inherited from Phalcon\Db\Adapter

Returns if nested transactions should use savepoints

public *string* **getNestedTransactionSavepointName** () inherited from Phalcon\Db\Adapter

Returns the savepoint name to use for nested transactions

public *Phalcon\Db\RawValue* **getDefaultIdValue** () inherited from Phalcon\Db\Adapter

Returns the default identity value to be inserted in an identity column

```
<?php

//Inserting a new robot with a valid default value for the column 'id'
$success = $connection->insert(
    "robots",
    array($connection->getDefaultIdValue(), "Astro Boy", 1952),
    array("id", "name", "year")
);
```

**public boolean supportSequences ()** inherited from Phalcon\Db\Adapter

Check whether the database system requires a sequence to produce auto-numeric values

**public boolean useExplicitIdValue ()** inherited from Phalcon\Db\Adapter

Check whether the database system requires an explicit value for identity columns

**public array getDescriptor ()** inherited from Phalcon\Db\Adapter

Return descriptor used to connect to the active database

**public string getConnectionId ()** inherited from Phalcon\Db\Adapter

Gets the active connection unique identifier

**public string getSQLStatement ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object

**public string getRealSQLStatement ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object without replace bound paramters

**public array getSQLVariables ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object

**public array getSQLBindTypes ()** inherited from Phalcon\Db\Adapter

Active SQL statement in the object

**public string getType ()** inherited from Phalcon\Db\Adapter

Returns type of database system the adapter is used for

**public string getDialectType ()** inherited from Phalcon\Db\Adapter

Returns the name of the dialect used

## 2.54.67 Class Phalcon\Db\Adapter\Pdo\Oracle

*extends abstract class Phalcon\Db\Adapter\Pdo*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\Db\AdapterInterface*

Specific functions for the Oracle database system

```
<?php
```

```
$config = array(
    "dbname" => "//localhost/dbname",
    "username" => "oracle",
    "password" => "oracle"
```

```
);  
  
$connection = new Phalcon\Db\Adapter\Pdo\Oracle($config);
```

## Methods

public *boolean* **connect** ([array \$descriptor])

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection.

public *Phalcon\Db\Column* [] **describeColumns** (string \$table, [string \$schema])

Returns an array of Phalcon\Db\Column objects describing a table <code>print\_r(\$connection->describeColumns("posts")); ?>

public *int* **lastInsertId** ([string \$sequenceName])

Returns the insert id for the auto\_increment/serial column inserted in the lastest executed SQL statement

```
<?php
```

```
//Inserting a new robot  
$success = $connection->insert(  
    "robots",  
    array("Astro Boy", 1952),  
    array("name", "year")  
);  
  
//Getting the generated id  
$id = $connection->lastInsertId();
```

public *boolean* **useExplicitIdValue** ()

Check whether the database system requires an explicit value for identity columns

public *Phalcon\Db\RawValue* **getDefaultIdValue** ()

Return the default identity value to insert in an identity column

public *boolean* **supportSequences** ()

Check whether the database system requires a sequence to produce auto-numeric values

public **\_\_construct** (array \$descriptor) inherited from Phalcon\Db\Adapter\Pdo

Constructor for Phalcon\Db\Adapter\Pdo

public *PDOStatement* **prepare** (string \$sqlStatement) inherited from Phalcon\Db\Adapter\Pdo

Returns a PDO prepared statement to be executed with 'executePrepared'

```
<?php
```

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');  
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

public *PDOStatement* **executePrepared** (*PDOStatement* \$statement, array \$placeholders, array \$dataTypes) inherited from Phalcon\Db\Adapter\Pdo

Executes a prepared statement binding. This function uses integer indexes starting from zero

```
<?php
```

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

*public Phalcon\Db\ResultInterface query (string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes])* inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server is returning rows

```
<?php
```

```
//Querying data
$resultset = $connection->query("SELECT * FROM robots WHERE type='mechanical'");
$resultset = $connection->query("SELECT * FROM robots WHERE type=?", array("mechanical"));
```

*public boolean execute (string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes])* inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server doesn't return any row

```
<?php
```

```
//Inserting data
$success = $connection->execute("INSERT INTO robots VALUES (1, 'Astro Boy')");
$success = $connection->execute("INSERT INTO robots VALUES (?, ?)", array(1, 'Astro Boy'));
```

*public int affectedRows ()* inherited from Phalcon\Db\Adapter\Pdo

Returns the number of affected rows by the lastest INSERT/UPDATE/DELETE executed in the database system

```
<?php
```

```
$connection->execute("DELETE FROM robots");
echo $connection->affectedRows(), ' were deleted';
```

*public boolean close ()* inherited from Phalcon\Db\Adapter\Pdo

Closes the active connection returning success. Phalcon automatically closes and destroys active connections when the request ends

*public string escapeIdentifier (string \$identifier)* inherited from Phalcon\Db\Adapter\Pdo

Escapes a column/table/schema name

```
<?php
```

```
$escapedTable = $connection->escapeIdentifier('robots');
$escapedTable = $connection->escapeIdentifier(array('store', 'robots'));
```

*public string escapeString (string \$str)* inherited from Phalcon\Db\Adapter\Pdo

Escapes a value to avoid SQL injections according to the active charset in the connection

```
<?php
```

```
$escapedStr = $connection->escapeString('some dangerous value');
```

public *array* **convertBoundParams** (*unknown* \$sqlStatement, *array* \$params) inherited from Phalcon\Db\Adapter\Pdo

Converts bound parameters such as :name: or ?1 into PDO bind params ?

<?php

```
print_r($connection->convertBoundParams('SELECT * FROM robots WHERE name = :name:', array('Bender')));
```

public *boolean* **begin** ([*boolean* \$nesting]) inherited from Phalcon\Db\Adapter\Pdo

Starts a transaction in the connection

public *boolean* **rollback** ([*boolean* \$nesting]) inherited from Phalcon\Db\Adapter\Pdo

Rollbacks the active transaction in the connection

public *boolean* **commit** ([*boolean* \$nesting]) inherited from Phalcon\Db\Adapter\Pdo

Commits the active transaction in the connection

public *int* **getTransactionLevel** () inherited from Phalcon\Db\Adapter\Pdo

Returns the current transaction nesting level

public *boolean* **isUnderTransaction** () inherited from Phalcon\Db\Adapter\Pdo

Checks whether the connection is under a transaction

<?php

```
$connection->begin();
var_dump($connection->isUnderTransaction()); //true
```

public *PDO* **getInternalHandler** () inherited from Phalcon\Db\Adapter\Pdo

Return internal PDO handler

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\Db\Adapter

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\Db\Adapter

Returns the internal event manager

public **setDialect** (*unknown* \$dialect) inherited from Phalcon\Db\Adapter

Sets the dialect used to produce the SQL

public *Phalcon\Db\ManagerInterface* **getDialect** () inherited from Phalcon\Db\Adapter

Returns internal dialect instance

public *array* **fetchOne** (*string* \$sqlQuery, [*int* \$fetchMode], [*unknown* \$placeholders]) inherited from Phalcon\Db\Adapter

Returns the first row in a SQL query result

<?php

```
//Getting first robot
$robot = $connection->fetchOne("SELECT * FROM robots");
print_r($robot);
```

```
//Getting first robot with associative indexes only
```

```
$robot = $connection->fetchOne("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
print_r($robot);
```

**public array fetchAll (string \$sqlQuery, [int \$fetchMode], [unknown \$placeholders])** inherited from Phalcon\Db\Adapter

Dumps the complete result of a query into an array

```
<?php
```

```
//Getting all robots with associative indexes only
$robots = $connection->fetchAll("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
foreach ($robots as $robot) {
    print_r($robot);
}

//Getting all robots that contains word "robot" withing the name
$robots = $connection->fetchAll("SELECT * FROM robots WHERE name LIKE :name",
    Phalcon\Db::FETCH_ASSOC,
    array('name' => '%robot%')
);
foreach($robots as $robot){
    print_r($robot);
}
```

**public boolean insert (string \$table, array \$values, [array \$fields], [array \$dataTypes])** inherited from Phalcon\Db\Adapter

Inserts data into a table using custom RBDM SQL syntax

```
<?php
```

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Next SQL sentence is sent to the database system
INSERT INTO `robots` ('name', 'year') VALUES ("Astro boy", 1952);
```

**public boolean update (string \$table, array \$fields, array \$values, [string \$whereCondition], [array \$dataTypes])** inherited from Phalcon\Db\Adapter

Updates data on a table using custom RBDM SQL syntax

```
<?php
```

```
//Updating existing robot
$success = $connection->update(
    "robots",
    array("name"),
    array("New Astro Boy"),
    "id = 101"
);

//Next SQL sentence is sent to the database system
UPDATE `robots` SET `name` = "Astro boy" WHERE id = 101
```

public *boolean delete* (*string \$tableName*, [*string \$whereCondition*], [*array \$placeholders*], [*array \$dataTypes*])  
inherited from Phalcon\Db\Adapter

Deletes data from a table using custom RBDM SQL syntax

<?php

```
//Deleting existing robot
$success = $connection->delete(
    "robots",
    "id = 101"
);

//Next SQL sentence is generated
DELETE FROM `robots` WHERE `id` = 101
```

public *string getColumnList* (*array \$columnList*) inherited from Phalcon\Db\Adapter

Gets a list of columns

public *string limit* (*string \$sqlQuery*, *int \$number*) inherited from Phalcon\Db\Adapter

Appends a LIMIT clause to \$sqlQuery argument

<?php

```
echo $connection->limit("SELECT * FROM robots", 5);
```

public *string tableExists* (*string \$tableName*, [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.table

<?php

```
var_dump($connection->tableExists("blog", "posts"));
```

public *string viewExists* (*string \$viewName*, [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.view

<?php

```
var_dump($connection->viewExists("active_users", "posts"));
```

public *string forUpdate* (*string \$sqlQuery*) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a FOR UPDATE clause

public *string sharedLock* (*string \$sqlQuery*) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a LOCK IN SHARE MODE clause

public *boolean createTable* (*string \$tableName*, *string \$schemaName*, [*array \$definition*]) inherited from Phalcon\Db\Adapter

Creates a table

public *boolean dropTable* (*string \$tableName*, [*string \$schemaName*], [*boolean \$ifExists*]) inherited from Phalcon\Db\Adapter

Drops a table from a schema/database

public *boolean createView* (*unknown \$viewName*, [*array \$definition*], [*string \$schemaName*]) inherited from Phalcon\Db\Adapter

Creates a view

`public boolean dropView (string $viewName, [string $schemaName], [boolean $IfExists])` inherited from Phalcon\Db\Adapter

Drops a view

`public boolean addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Adds a column to a table

`public boolean modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Modifies a table column based on a definition

`public boolean dropColumn (string $tableName, string $schemaName, string $columnName)` inherited from Phalcon\Db\Adapter

Drops a column from a table

`public boolean addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds an index to a table

`public boolean dropIndex (string $tableName, string $schemaName, string $indexName)` inherited from Phalcon\Db\Adapter

Drop an index from a table

`public boolean addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds a primary key to a table

`public boolean dropPrimaryKey (string $tableName, string $schemaName)` inherited from Phalcon\Db\Adapter

Drops a table's primary key

`public boolean true addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)` inherited from Phalcon\Db\Adapter

Adds a foreign key to a table

`public boolean true dropForeignKey (string $tableName, string $schemaName, string $referenceName)` inherited from Phalcon\Db\Adapter

Drops a foreign key from a table

`public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Returns the SQL column definition from a column

`public array listTables ([string $schemaName])` inherited from Phalcon\Db\Adapter

List all tables on a database

<?php

```
print_r($connection->listTables("blog"));
```

public *array* **listViews** (*[string \$schemaName]*) inherited from Phalcon\Db\Adapter

List all views on a database

<?php

```
print_r($connection->listViews("blog")); ?>
```

public *Phalcon\Db\Index* [] **describeIndexes** (*string \$table, [string \$schema]*) inherited from Phalcon\Db\Adapter

Lists table indexes

<?php

```
print_r($connection->describeIndexes('robots_parts'));
```

public *Phalcon\Db\Reference* [] **describeReferences** (*string \$table, [string \$schema]*) inherited from Phalcon\Db\Adapter

Lists table references

<?php

```
print_r($connection->describeReferences('robots_parts'));
```

public *array* **tableOptions** (*string \$tableName, [string \$schemaName]*) inherited from Phalcon\Db\Adapter

Gets creation options from a table

<?php

```
print_r($connection->tableOptions('robots'));
```

public *boolean* **createSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Creates a new savepoint

public *boolean* **releaseSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Releases given savepoint

public *boolean* **rollbackSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Rollbacks given savepoint

public *Phalcon\Db\AdapterInterface* **setNestedTransactionsWithSavepoints** (*boolean \$nestedTransactionsWithSavepoints*) inherited from Phalcon\Db\Adapter

Set if nested transactions should use savepoints

public *boolean* **isNestedTransactionsWithSavepoints** () inherited from Phalcon\Db\Adapter

Returns if nested transactions should use savepoints

public *string* **getNestedTransactionSavepointName** () inherited from Phalcon\Db\Adapter

Returns the savepoint name to use for nested transactions

public *array* **getDescriptor** () inherited from Phalcon\Db\Adapter

Return descriptor used to connect to the active database

public *string* **getConnectionId** () inherited from Phalcon\Db\Adapter

Gets the active connection unique identifier

**public string getSQLStatement ()** inherited from Phalcon\Db\Adapter  
 Active SQL statement in the object

**public string getRealSQLStatement ()** inherited from Phalcon\Db\Adapter  
 Active SQL statement in the object without replace bound paramters

**public array getSQLVariables ()** inherited from Phalcon\Db\Adapter  
 Active SQL statement in the object

**public array getSQLBindTypes ()** inherited from Phalcon\Db\Adapter  
 Active SQL statement in the object

**public string getType ()** inherited from Phalcon\Db\Adapter  
 Returns type of database system the adapter is used for

**public string getDialectType ()** inherited from Phalcon\Db\Adapter  
 Returns the name of the dialect used

## 2.54.68 Class Phalcon\Db\Adapter\Pdo\Postgresql

*extends abstract class Phalcon\Db\Adapter\Pdo*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\Db\AdapterInterface*

Specific functions for the Postgresql database system

<?php

```
$config = array(
    "host" => "192.168.0.11",
    "dbname" => "blog",
    "username" => "postgres",
    "password" => ""
);

$connection = new Phalcon\Db\Adapter\Pdo\Postgresql($config);
```

### Methods

**public boolean connect ([array \$descriptor])**

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection. Support set search\_path after connected if schema is specified in config.

**public Phalcon\Db\Column [] describeColumns (string \$table, [string \$schema])**

Returns an array of Phalcon\Db\Column objects describing a table <code>print\_r(\$connection->describeColumns("posts")); ?>

**public boolean useExplicitIdValue ()**

Check whether the database system requires an explicit value for identity columns

**public Phalcon\Db\RawValue getDefaultIdValue ()**

Return the default identity value to insert in an identity column

**public boolean supportSequences ()**

Check whether the database system requires a sequence to produce auto-numeric values

public **\_\_construct** (*array \$descriptor*) inherited from Phalcon\Db\Adapter\Pdo

Constructor for Phalcon\Db\Adapter\Pdo

public *PDOStatement* **prepare** (*string \$sqlStatement*) inherited from Phalcon\Db\Adapter\Pdo

Returns a PDO prepared statement to be executed with ‘executePrepared’

<?php

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

public *PDOStatement* **executePrepared** (*PDOStatement \$statement, array \$placeholders, array \$dataTypes*) inherited from Phalcon\Db\Adapter\Pdo

Executes a prepared statement binding. This function uses integer indexes starting from zero

<?php

```
$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

public *Phalcon\Db\ResultInterface* **query** (*string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes]*) inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server is returning rows

<?php

```
//Querying data
$resultset = $connection->query("SELECT * FROM robots WHERE type='mechanical'");
$resultset = $connection->query("SELECT * FROM robots WHERE type=?", array("mechanical"));
```

public *boolean execute* (*string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes]*) inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server doesn’t return any row

<?php

```
//Inserting data
$success = $connection->execute("INSERT INTO robots VALUES (1, 'Astro Boy')");
$success = $connection->execute("INSERT INTO robots VALUES (?, ?)", array(1, 'Astro Boy'));
```

public *int affectedRows* () inherited from Phalcon\Db\Adapter\Pdo

Returns the number of affected rows by the lastest INSERT/UPDATE/DELETE executed in the database system

<?php

```
$connection->execute("DELETE FROM robots");
echo $connection->affectedRows(), ' were deleted';
```

public *boolean close* () inherited from Phalcon\Db\Adapter\Pdo

Closes the active connection returning success. Phalcon automatically closes and destroys active connections when the request ends

**public string escapeIdentifier (string \$identifier)** inherited from Phalcon\Db\Adapter\Pdo  
Escapes a column/table/schema name

<?php

```
$escapedTable = $connection->escapeIdentifier('robots');
$escapedTable = $connection->escapeIdentifier(array('store', 'robots'));
```

**public string escapeString (string \$str)** inherited from Phalcon\Db\Adapter\Pdo  
Escapes a value to avoid SQL injections according to the active charset in the connection

<?php

```
$escapedStr = $connection->escapeString('some dangerous value');
```

**public array convertBoundParams (unknown \$sqlStatement, array \$params)** inherited from Phalcon\Db\Adapter\Pdo

Converts bound parameters such as :name: or ?1 into PDO bind params ?

<?php

```
print_r($connection->convertBoundParams('SELECT * FROM robots WHERE name = :name:', array('Bender')));
```

**public int lastInsertId ([string \$sequenceName])** inherited from Phalcon\Db\Adapter\Pdo

Returns the insert id for the auto\_increment/serial column inserted in the lastest executed SQL statement

<?php

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);
```

```
//Getting the generated id
$id = $connection->lastInsertId();
```

**public boolean begin ([boolean \$nesting])** inherited from Phalcon\Db\Adapter\Pdo

Starts a transaction in the connection

**public boolean rollback ([boolean \$nesting])** inherited from Phalcon\Db\Adapter\Pdo

Rollbacks the active transaction in the connection

**public boolean commit ([boolean \$nesting])** inherited from Phalcon\Db\Adapter\Pdo

Commits the active transaction in the connection

**public int getTransactionLevel ()** inherited from Phalcon\Db\Adapter\Pdo

Returns the current transaction nesting level

**public boolean isUnderTransaction ()** inherited from Phalcon\Db\Adapter\Pdo

Checks whether the connection is under a transaction

<?php

```
$connection->begin();
var_dump($connection->isUnderTransaction()); //true
```

public *PDO* **getInternalHandler** () inherited from Phalcon\Db\Adapter\Pdo  
Return internal PDO handler

```
public setEventsManager (Phalcon\Events\ManagerInterface $eventsManager) inherited from Phalcon\Db\Adapter
```

Sets the event manager

```
public Phalcon\Events\ManagerInterface getEventsManager () inherited from Phalcon\Db\Adapter
```

Returns the internal event manager

```
public setDialect (unknown $dialect) inherited from Phalcon\Db\Adapter
```

Sets the dialect used to produce the SQL

```
public Phalcon\Db\ManagerInterface getDialect () inherited from Phalcon\Db\Adapter
```

Returns internal dialect instance

```
public array fetchOne (string $sqlQuery, [int $fetchMode], [unknown $placeholders]) inherited from Phalcon\Db\Adapter
```

Returns the first row in a SQL query result

```
<?php

//Getting first robot
$robot = $connection->fetchOne("SELECT * FROM robots");
print_r($robot);

//Getting first robot with associative indexes only
$robot = $connection->fetchOne("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
print_r($robot);
```

```
public array fetchAll (string $sqlQuery, [int $fetchMode], [unknown $placeholders]) inherited from Phalcon\Db\Adapter
```

Dumps the complete result of a query into an array

```
<?php

//Getting all robots with associative indexes only
$robots = $connection->fetchAll("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
foreach ($robots as $robot) {
    print_r($robot);
}

//Getting all robots that contains word "robot" withing the name
$robots = $connection->fetchAll("SELECT * FROM robots WHERE name LIKE :name",
    Phalcon\Db::FETCH_ASSOC,
    array('name' => '%robot%')
);
foreach($robots as $robot){
    print_r($robot);
}
```

```
public boolean insert (string $table, array $values, [array $fields], [array $dataTypes]) inherited from Phalcon\Db\Adapter
```

Inserts data into a table using custom RBDM SQL syntax

```
<?php

//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Next SQL sentence is sent to the database system
INSERT INTO `robots` ('name', 'year') VALUES ("Astro boy", 1952);

public boolean update (string $table, array $fields, array $values, [string $whereCondition], [array $dataTypes]) inherited from Phalcon\Db\Adapter
```

Updates data on a table using custom RBDM SQL syntax

```
<?php

//Updating existing robot
$success = $connection->update(
    "robots",
    array("name"),
    array("New Astro Boy"),
    "id = 101"
);

//Next SQL sentence is sent to the database system
UPDATE `robots` SET `name` = "Astro boy" WHERE id = 101
```

```
public boolean delete (string $table, [string $whereCondition], [array $placeholders], [array $dataTypes]) inherited from Phalcon\Db\Adapter
```

Deletes data from a table using custom RBDM SQL syntax

```
<?php

//Deleting existing robot
$success = $connection->delete(
    "robots",
    "id = 101"
);

//Next SQL sentence is generated
DELETE FROM `robots` WHERE `id` = 101
```

public string getColumnList (array \$columnList) inherited from Phalcon\Db\Adapter

Gets a list of columns

public string limit (string \$sqlQuery, int \$number) inherited from Phalcon\Db\Adapter

Appends a LIMIT clause to \$sqlQuery argument

```
<?php

echo $connection->limit("SELECT * FROM robots", 5);

public string tableExists (string $tableName, [string $schemaName]) inherited from Phalcon\Db\Adapter
```

Generates SQL checking for the existence of a schema.table

<?php

```
var_dump($connection->tableExists("blog", "posts"));
```

public *string* **viewExists** (*string* \$viewName, [*string* \$schemaName]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.view

<?php

```
var_dump($connection->viewExists("active_users", "posts"));
```

public *string* **forUpdate** (*string* \$sqlQuery) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a FOR UPDATE clause

public *string* **sharedLock** (*string* \$sqlQuery) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a LOCK IN SHARE MODE clause

public *boolean* **createTable** (*string* \$tableName, *string* \$schemaName, *array* \$definition) inherited from Phalcon\Db\Adapter

Creates a table

public *boolean* **dropTable** (*string* \$tableName, [*string* \$schemaName], [*boolean* \$IfExists]) inherited from Phalcon\Db\Adapter

Drops a table from a schema/database

public *boolean* **createView** (*unknown* \$viewName, *array* \$definition, [*string* \$schemaName]) inherited from Phalcon\Db\Adapter

Creates a view

public *boolean* **dropView** (*string* \$viewName, [*string* \$schemaName], [*boolean* \$IfExists]) inherited from Phalcon\Db\Adapter

Drops a view

public *boolean* **addColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column) inherited from Phalcon\Db\Adapter

Adds a column to a table

public *boolean* **modifyColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column) inherited from Phalcon\Db\Adapter

Modifies a table column based on a definition

public *boolean* **dropColumn** (*string* \$tableName, *string* \$schemaName, *string* \$columnName) inherited from Phalcon\Db\Adapter

Drops a column from a table

public *boolean* **addIndex** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\IndexInterface* \$index) inherited from Phalcon\Db\Adapter

Adds an index to a table

public *boolean* **dropIndex** (*string* \$tableName, *string* \$schemaName, *string* \$indexName) inherited from Phalcon\Db\Adapter

Drop an index from a table

`public boolean addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\Adapter

Adds a primary key to a table

`public boolean dropPrimaryKey (string $tableName, string $schemaName)` inherited from Phalcon\Db\Adapter

Drops a table's primary key

`public boolean true addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)` inherited from Phalcon\Db\Adapter

Adds a foreign key to a table

`public boolean true dropForeignKey (string $tableName, string $schemaName, string $referenceName)` inherited from Phalcon\Db\Adapter

Drops a foreign key from a table

`public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\Adapter

Returns the SQL column definition from a column

`public array listTables ([string $schemaName])` inherited from Phalcon\Db\Adapter

List all tables on a database

```
<?php
```

```
print_r($connection->listTables("blog"));
```

`public array listViews ([string $schemaName])` inherited from Phalcon\Db\Adapter

List all views on a database

```
<?php
```

```
print_r($connection->listViews("blog")); ?>
```

`public Phalcon\Db\Index [] describeIndexes (string $table, [string $schema])` inherited from Phalcon\Db\Adapter

Lists table indexes

```
<?php
```

```
print_r($connection->describeIndexes('robots_parts'));
```

`public Phalcon\Db\Reference [] describeReferences (string $table, [string $schema])` inherited from Phalcon\Db\Adapter

Lists table references

```
<?php
```

```
print_r($connection->describeReferences('robots_parts'));
```

`public array tableOptions (string $tableName, [string $schemaName])` inherited from Phalcon\Db\Adapter

Gets creation options from a table

```
<?php  
  
print_r($connection->tableOptions('robots'));  
  
public boolean createSavepoint (string $name) inherited from Phalcon\Db\Adapter  
Creates a new savepoint  
public boolean releaseSavepoint (string $name) inherited from Phalcon\Db\Adapter  
Releases given savepoint  
public boolean rollbackSavepoint (string $name) inherited from Phalcon\Db\Adapter  
Rollbacks given savepoint  
public Phalcon\Db\AdapterInterface setNestedTransactionsWithSavepoints (boolean  
$nestedTransactionsWithSavepoints) inherited from Phalcon\Db\Adapter  
Set if nested transactions should use savepoints  
public boolean isNestedTransactionsWithSavepoints () inherited from Phalcon\Db\Adapter  
Returns if nested transactions should use savepoints  
public string getNestedTransactionSavepointName () inherited from Phalcon\Db\Adapter  
Returns the savepoint name to use for nested transactions  
public array getDescriptor () inherited from Phalcon\Db\Adapter  
Return descriptor used to connect to the active database  
public string getConnectionId () inherited from Phalcon\Db\Adapter  
Gets the active connection unique identifier  
public string getSQLStatement () inherited from Phalcon\Db\Adapter  
Active SQL statement in the object  
public string getRealSQLStatement () inherited from Phalcon\Db\Adapter  
Active SQL statement in the object without replace bound paramters  
public array getSQLVariables () inherited from Phalcon\Db\Adapter  
Active SQL statement in the object  
public array getSQLBindTypes () inherited from Phalcon\Db\Adapter  
Active SQL statement in the object  
public string getType () inherited from Phalcon\Db\Adapter  
Returns type of database system the adapter is used for  
public string getDialectType () inherited from Phalcon\Db\Adapter  
Returns the name of the dialect used
```

## 2.54.69 Class Phalcon\Db\Adapter\Pdo\Sqlite

*extends* abstract class Phalcon\Db\Adapter\Pdo  
*implements* Phalcon\Events\EventsAwareInterface, Phalcon\Db\AdapterInterface

Specific functions for the Sqlite database system

```
<?php

$config = array(
    "dbname" => "/tmp/test.sqlite"
);

$connection = new Phalcon\Db\Adapter\Pdo\Sqlite($config);
```

## Methods

**public boolean connect ([array \$descriptor])**

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection.

**public Phalcon\Db\Column [] describeColumns (string \$table, [string \$schema])**

Returns an array of Phalcon\Db\Column objects describing a table

```
<?php

print_r($connection->describeColumns("posts")); ?>
```

**public Phalcon\Db\Index [] describeIndexes (string \$table, [string \$schema])**

Lists table indexes

**public Phalcon\Db\Reference [] describeReferences (string \$table, [string \$schema])**

Lists table references

**public boolean useExplicitIdValue ()**

Check whether the database system requires an explicit value for identity columns

**public \_\_construct (array \$descriptor)** inherited from Phalcon\Db\Adapter\Pdo

Constructor for Phalcon\Db\Adapter\Pdo

**public PDOStatement prepare (string \$sqlStatement)** inherited from Phalcon\Db\Adapter\Pdo

Returns a PDO prepared statement to be executed with ‘executePrepared’

```
<?php

$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

**public PDOStatement executePrepared (PDOStatement \$statement, array \$placeholders, array \$dataTypes)** inherited from Phalcon\Db\Adapter\Pdo

Executes a prepared statement binding. This function uses integer indexes starting from zero

```
<?php

$statement = $connection->prepare('SELECT * FROM robots WHERE name = :name');
$pdoResult = $connection->executePrepared($statement, array('name' => 'Voltron'));
```

**public Phalcon\Db\ResultInterface query (string \$sqlStatement, [unknown \$placeholders], [unknown \$dataTypes])** inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server is returning rows

```
<?php

//Querying data
$resultset = $connection->query("SELECT * FROM robots WHERE type='mechanical'");
$resultset = $connection->query("SELECT * FROM robots WHERE type=?", array("mechanical"));
```

public *boolean execute* (*string* \$sqlStatement, [*unknown* \$placeholders], [*unknown* \$dataTypes]) inherited from Phalcon\Db\Adapter\Pdo

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server doesn't return any row

```
<?php

//Inserting data
$success = $connection->execute("INSERT INTO robots VALUES (1, 'Astro Boy')");
$success = $connection->execute("INSERT INTO robots VALUES (?, ?)", array(1, 'Astro Boy'));
```

public *int affectedRows* () inherited from Phalcon\Db\Adapter\Pdo

Returns the number of affected rows by the lastest INSERT/UPDATE/DELETE executed in the database system

```
<?php

$connection->execute("DELETE FROM robots");
echo $connection->affectedRows(), ' were deleted';
```

public *boolean close* () inherited from Phalcon\Db\Adapter\Pdo

Closes the active connection returning success. Phalcon automatically closes and destroys active connections when the request ends

public *string escapeIdentifier* (*string* \$identifier) inherited from Phalcon\Db\Adapter\Pdo

Escapes a column/table/schema name

```
<?php

$escapedTable = $connection->escapeIdentifier('robots');
$escapedTable = $connection->escapeIdentifier(array('store', 'robots'));
```

public *string escapeString* (*string* \$str) inherited from Phalcon\Db\Adapter\Pdo

Escapes a value to avoid SQL injections according to the active charset in the connection

```
<?php
```

```
$escapedStr = $connection->escapeString('some dangerous value');
```

public *array convertBoundParams* (*unknown* \$sqlStatement, *array* \$params) inherited from Phalcon\Db\Adapter\Pdo

Converts bound parameters such as :name: or ?1 into PDO bind params ?

```
<?php
```

```
print_r($connection->convertBoundParams('SELECT * FROM robots WHERE name = :name:', array('Bender')));
```

**public int lastInsertId ([string \$sequenceName])** inherited from Phalcon\Db\Adapter\Pdo

Returns the insert id for the auto\_increment/serial column inserted in the lastest executed SQL statement

<?php

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Getting the generated id
$id = $connection->lastInsertId();
```

**public boolean begin ([boolean \$nesting])** inherited from Phalcon\Db\Adapter\Pdo

Starts a transaction in the connection

**public boolean rollback ([boolean \$nesting])** inherited from Phalcon\Db\Adapter\Pdo

Rollbacks the active transaction in the connection

**public boolean commit ([boolean \$nesting])** inherited from Phalcon\Db\Adapter\Pdo

Commits the active transaction in the connection

**public int getTransactionLevel ()** inherited from Phalcon\Db\Adapter\Pdo

Returns the current transaction nesting level

**public boolean isUnderTransaction ()** inherited from Phalcon\Db\Adapter\Pdo

Checks whether the connection is under a transaction

<?php

```
$connection->begin();
var_dump($connection->isUnderTransaction()); //true
```

**public PDO getInternalHandler ()** inherited from Phalcon\Db\Adapter\Pdo

Return internal PDO handler

**public setEventsManager (Phalcon\Events\ManagerInterface \$eventsManager)** inherited from Phalcon\Db\Adapter

Sets the event manager

**public Phalcon\Events\ManagerInterface getEventsManager ()** inherited from Phalcon\Db\Adapter

Returns the internal event manager

**public setDialect (unknown \$dialect)** inherited from Phalcon\Db\Adapter

Sets the dialect used to produce the SQL

**public Phalcon\Db\DialetInterface getDialect ()** inherited from Phalcon\Db\Adapter

Returns internal dialect instance

**public array fetchOne (string \$sqlQuery, [int \$fetchMode], [unknown \$placeholders])** inherited from Phalcon\Db\Adapter

Returns the first row in a SQL query result

```
<?php
```

```
//Getting first robot
$robot = $connection->fetchOne("SELECT * FROM robots");
print_r($robot);

//Getting first robot with associative indexes only
$robot = $connection->fetchOne("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
print_r($robot);
```

public *array* **fetchAll** (*string* \$sqlQuery, [*int* \$fetchMode], [*unknown* \$placeholders]) inherited from Phalcon\Db\Adapter

Dumps the complete result of a query into an array

```
<?php
```

```
//Getting all robots with associative indexes only
$robots = $connection->fetchAll("SELECT * FROM robots", Phalcon\Db::FETCH_ASSOC);
foreach ($robots as $robot) {
    print_r($robot);
}

//Getting all robots that contains word "robot" withing the name
$robots = $connection->fetchAll("SELECT * FROM robots WHERE name LIKE :name",
    Phalcon\Db::FETCH_ASSOC,
    array('name' => '%robot%')
);
foreach($robots as $robot){
    print_r($robot);
}
```

public *boolean* **insert** (*string* \$table, *array* \$values, [*array* \$fields], [*array* \$dataTypes]) inherited from Phalcon\Db\Adapter

Inserts data into a table using custom RBDM SQL syntax

```
<?php
```

```
//Inserting a new robot
$success = $connection->insert(
    "robots",
    array("Astro Boy", 1952),
    array("name", "year")
);

//Next SQL sentence is sent to the database system
INSERT INTO `robots` ('name', 'year') VALUES ("Astro boy", 1952);
```

public *boolean* **update** (*string* \$table, *array* \$fields, *array* \$values, [*string* \$whereCondition], [*array* \$dataTypes]) inherited from Phalcon\Db\Adapter

Updates data on a table using custom RBDM SQL syntax

```
<?php
```

```
//Updating existing robot
$success = $connection->update(
    "robots",
    array("name"),
```

```

array("New Astro Boy"),
      "id = 101"
);

//Next SQL sentence is sent to the database system
UPDATE `robots` SET `name` = "Astro boy" WHERE id = 101

public boolean delete (string $table, [string $whereCondition], [array $placeholders], [array $dataTypes])
inherited from Phalcon\Db\Adapter

Deletes data from a table using custom RBDM SQL syntax

<?php

//Deleting existing robot
$success = $connection->delete(
    "robots",
    "id = 101"
);

//Next SQL sentence is generated
DELETE FROM `robots` WHERE `id` = 101

public string getColumnList (array $columnList) inherited from Phalcon\Db\Adapter

Gets a list of columns

public string limit (string $sqlQuery, int $number) inherited from Phalcon\Db\Adapter

Appends a LIMIT clause to $sqlQuery argument

<?php

echo $connection->limit("SELECT * FROM robots", 5);

public string tableExists (string $tableName, [string $schemaName]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.table

<?php

var_dump($connection->tableExists("blog", "posts"));

public string viewExists (string $viewName, [string $schemaName]) inherited from Phalcon\Db\Adapter

Generates SQL checking for the existence of a schema.view

<?php

var_dump($connection->viewExists("active_users", "posts"));

public string forUpdate (string $sqlQuery) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a FOR UPDATE clause

public string sharedLock (string $sqlQuery) inherited from Phalcon\Db\Adapter

Returns a SQL modified with a LOCK IN SHARE MODE clause

public boolean createTable (string $tableName, string $schemaName, array $definition) inherited from
Phalcon\Db\Adapter

Creates a table

```

public *boolean* **dropTable** (*string* \$tableName, [*string* \$schemaName], [*boolean* \$IfExists]) inherited from Phalcon\Db\Adapter

Drops a table from a schema/database

public *boolean* **createView** (*unknown* \$viewName, *array* \$definition, [*string* \$schemaName]) inherited from Phalcon\Db\Adapter

Creates a view

public *boolean* **dropView** (*string* \$viewName, [*string* \$schemaName], [*boolean* \$IfExists]) inherited from Phalcon\Db\Adapter

Drops a view

public *boolean* **addColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column) inherited from Phalcon\Db\Adapter

Adds a column to a table

public *boolean* **modifyColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column) inherited from Phalcon\Db\Adapter

Modifies a table column based on a definition

public *boolean* **dropColumn** (*string* \$tableName, *string* \$schemaName, *string* \$columnName) inherited from Phalcon\Db\Adapter

Drops a column from a table

public *boolean* **addIndex** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\IndexInterface* \$index) inherited from Phalcon\Db\Adapter

Adds an index to a table

public *boolean* **dropIndex** (*string* \$tableName, *string* \$schemaName, *string* \$indexName) inherited from Phalcon\Db\Adapter

Drop an index from a table

public *boolean* **addPrimaryKey** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\IndexInterface* \$index) inherited from Phalcon\Db\Adapter

Adds a primary key to a table

public *boolean* **dropPrimaryKey** (*string* \$tableName, *string* \$schemaName) inherited from Phalcon\Db\Adapter

Drops a table's primary key

public *boolean* **true addForeignKey** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ReferenceInterface* \$reference) inherited from Phalcon\Db\Adapter

Adds a foreign key to a table

public *boolean* **true dropForeignKey** (*string* \$tableName, *string* \$schemaName, *string* \$referenceName) inherited from Phalcon\Db\Adapter

Drops a foreign key from a table

public *string* **getColumnDefinition** (*Phalcon\Db\ColumnInterface* \$column) inherited from Phalcon\Db\Adapter

Returns the SQL column definition from a column

public *array* **listTables** ([*string* \$schemaName]) inherited from Phalcon\Db\Adapter

List all tables on a database

```
<?php
    print_r($connection->listTables("blog"));
```

public *array* **listViews** (*[string \$schemaName]*) inherited from Phalcon\Db\Adapter

List all views on a database

```
<?php
    print_r($connection->listViews("blog")); ?>
```

public *array* **tableOptions** (*string \$tableName, [string \$schemaName]*) inherited from Phalcon\Db\Adapter

Gets creation options from a table

```
<?php
    print_r($connection->tableOptions('robots'));
```

public *boolean* **createSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Creates a new savepoint

public *boolean* **releaseSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Releases given savepoint

public *boolean* **rollbackSavepoint** (*string \$name*) inherited from Phalcon\Db\Adapter

Rollbacks given savepoint

public *Phalcon\Db\AdapterInterface* **setNestedTransactionsWithSavepoints** (*boolean \$nestedTransactionsWithSavepoints*) inherited from Phalcon\Db\Adapter

Set if nested transactions should use savepoints

public *boolean* **isNestedTransactionsWithSavepoints** () inherited from Phalcon\Db\Adapter

Returns if nested transactions should use savepoints

public *string* **getNestedTransactionSavepointName** () inherited from Phalcon\Db\Adapter

Returns the savepoint name to use for nested transactions

public *Phalcon\Db\RawValue* **getDefaultIdValue** () inherited from Phalcon\Db\Adapter

Returns the default identity value to be inserted in an identity column

```
<?php
```

```
//Inserting a new robot with a valid default value for the column 'id'
$success = $connection->insert(
    "robots",
    array($connection->getDefaultIdValue(), "Astro Boy", 1952),
    array("id", "name", "year")
);
```

public *boolean* **supportSequences** () inherited from Phalcon\Db\Adapter

Check whether the database system requires a sequence to produce auto-numeric values

public *array* **getDescriptor** () inherited from Phalcon\Db\Adapter

Return descriptor used to connect to the active database

`public string getConnectionId ()` inherited from Phalcon\Db\Adapter

Gets the active connection unique identifier

`public string getSQLStatement ()` inherited from Phalcon\Db\Adapter

Active SQL statement in the object

`public string getRealSQLStatement ()` inherited from Phalcon\Db\Adapter

Active SQL statement in the object without replace bound parameters

`public array getSQLVariables ()` inherited from Phalcon\Db\Adapter

Active SQL statement in the object

`public array getSQLBindTypes ()` inherited from Phalcon\Db\Adapter

Active SQL statement in the object

`public string getType ()` inherited from Phalcon\Db\Adapter

Returns type of database system the adapter is used for

`public string getDialectType ()` inherited from Phalcon\Db\Adapter

Returns the name of the dialect used

## 2.54.70 Class Phalcon\Db\Column

*implements Phalcon\Db\ColumnInterface*

Allows to define columns to be used on create or alter table operations

<?php

```
use Phalcon\Db\Column as Column;

//column definition
$column = new Column("id", array(
    "type" => Column::TYPE_INTEGER,
    "size" => 10,
    "unsigned" => true,
    "notNull" => true,
    "autoIncrement" => true,
    "first" => true
));
//add column to existing table
$connection->addColumn("robots", null, $column);
```

### Constants

`integer TYPE_INTEGER`

`integer TYPE_DATE`

`integer TYPE_VARCHAR`

`integer TYPE_DECIMAL`

```
integer TYPE_DATETIME  
integer TYPE_CHAR  
integer TYPE_TEXT  
integer TYPE_FLOAT  
integer TYPE_BOOLEAN  
integer TYPE_DOUBLE  
integer BIND_PARAM_NULL  
integer BIND_PARAM_INT  
integer BIND_PARAM_STR  
integer BIND_PARAM_BOOL  
integer BIND_PARAM_DECIMAL  
integer BIND_SKIP
```

## Methods

public **\_\_construct** (*string* \$columnName, *array* \$definition)

Phalcon\Db\Column constructor

public *string* **getSchemaName** ()

Returns schema's table related to column

public *string* **getName** ()

Returns column name

public *int* **getType** ()

Returns column type

public *int* **getSize** ()

Returns column size

public *int* **getScale** ()

Returns column scale

public *boolean* **isUnsigned** ()

Returns true if number column is unsigned

public *boolean* **isNotNull** ()

Not null

public *boolean* **isPrimary** ()

Column is part of the primary key?

public *boolean* **isAutoIncrement** ()

Auto-Increment

public *boolean* **isNumeric** ()

Check whether column have an numeric type

```
public boolean isFirst ()  
Check whether column have first position in table  
public string getAfterPosition ()  
Check whether field absolute to position in table  
public int getBindType ()  
Returns the type of bind handling  
public static Phalcon\Db\Column __set_state ([unknown $properties])  
Restores the internal state of a Phalcon\Db\Column object
```

## 2.54.71 Abstract class Phalcon\Db\Dialect

*implements Phalcon\Db\DialectInterface*

This is the base class to each database dialect. This implements common methods to transform intermediate code into its RDBM related syntax

### Methods

```
public string limit (string $sqlQuery, int $number)
```

Generates the SQL for LIMIT clause

```
<?php
```

```
$sql = $dialect->limit('SELECT * FROM robots', 10);  
echo $sql; // SELECT * FROM robots LIMIT 10
```

```
public string forUpdate (string $sqlQuery)
```

Returns a SQL modified with a FOR UPDATE clause

```
<?php
```

```
$sql = $dialect->forUpdate('SELECT * FROM robots');  
echo $sql; // SELECT * FROM robots FOR UPDATE
```

```
public string sharedLock (string $sqlQuery)
```

Returns a SQL modified with a LOCK IN SHARE MODE clause

```
<?php
```

```
$sql = $dialect->sharedLock('SELECT * FROM robots');  
echo $sql; // SELECT * FROM robots LOCK IN SHARE MODE
```

```
public string getColumnList (array $columnList)
```

Gets a list of columns with escaped identifiers

```
<?php
```

```
echo $dialect->getColumnList(array('column1', 'column'));
```

`public string getSqlExpression (array $expression, [string $escapeChar])`

Transforms an intermediate representation for a expression into a database system valid expression

`public string getSqlTable (array $table, [string $escapeChar])`

Transform an intermediate representation for a schema/table into a database system valid expression

`public string select (array $definition)`

Builds a SELECT statement

`public boolean supportsSavepoints ()`

Checks whether the platform supports savepoints

`public boolean supportsReleaseSavepoints ()`

Checks whether the platform supports releasing savepoints.

`public string createSavepoint (string $name)`

Generate SQL to create a new savepoint

`public string releaseSavepoint (string $name)`

Generate SQL to release a savepoint

`public string rollbackSavepoint (string $name)`

Generate SQL to rollback a savepoint

`abstract public getColumnDefinition (Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\DialectInterface

Gets the column name in MySQL

`abstract public string addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\DialectInterface

Generates SQL to add a column to a table

`abstract public string modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)` inherited from Phalcon\Db\DialectInterface

Generates SQL to modify a column in a table

`abstract public string dropColumn (string $tableName, string $schemaName, string $columnName)` inherited from Phalcon\Db\DialectInterface

Generates SQL to delete a column from a table

`abstract public string addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\DialectInterface

Generates SQL to add an index to a table

`abstract public string dropIndex (string $tableName, string $schemaName, string $indexName)` inherited from Phalcon\Db\DialectInterface

Generates SQL to delete an index from a table

`abstract public string addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)` inherited from Phalcon\Db\DialectInterface

Generates SQL to add the primary key to a table

`abstract public string dropPrimaryKey (string $tableName, string $schemaName)` inherited from Phalcon\Db\DialectInterface

Generates SQL to delete primary key from a table

```
abstract public string addForeignKey (string $tableName, string $schemaName,  
Phalcon\Db\ReferenceInterface $reference) inherited from Phalcon\Db\DialectInterface
```

Generates SQL to add an index to a table

```
abstract public string dropForeignKey (string $tableName, string $schemaName, string $referenceName)  
inherited from Phalcon\Db\DialectInterface
```

Generates SQL to delete a foreign key from a table

```
abstract public string createTable (string $tableName, string $schemaName, array $definition) inherited  
from Phalcon\Db\DialectInterface
```

Generates SQL to create a table

```
abstract public string dropTable (string $tableName, string $schemaName) inherited from  
Phalcon\Db\DialectInterface
```

Generates SQL to drop a table

```
abstract public string createView (string $viewName, array $definition, string $schemaName) inherited  
from Phalcon\Db\DialectInterface
```

Generates SQL to create a view

```
abstract public string dropView (string $viewName, string $schemaName, [unknown $IfExists]) inherited  
from Phalcon\Db\DialectInterface
```

Generates SQL to drop a view

```
abstract public string tableExists (string $tableName, [string $schemaName]) inherited from  
Phalcon\Db\DialectInterface
```

Generates SQL checking for the existence of a schema.table

```
abstract public string viewExists (string $viewName, [string $schemaName]) inherited from  
Phalcon\Db\DialectInterface
```

Generates SQL checking for the existence of a schema.view

```
abstract public string describeColumns (string $table, [string $schema]) inherited from  
Phalcon\Db\DialectInterface
```

Generates SQL to describe a table

```
abstract public array listTables ([string $schemaName]) inherited from Phalcon\Db\DialectInterface
```

List all tables on database

```
abstract public array listViews ([string $schemaName]) inherited from Phalcon\Db\DialectInterface
```

List all views on database

```
abstract public string describeIndexes (string $table, [string $schema]) inherited from  
Phalcon\Db\DialectInterface
```

Generates SQL to query indexes on a table

```
abstract public string describeReferences (string $table, [string $schema]) inherited from  
Phalcon\Db\DialectInterface
```

Generates SQL to query foreign keys on a table

```
abstract public string tableOptions (string $table, [string $schema]) inherited from  
Phalcon\Db\DialectInterface
```

Generates the SQL to describe the table creation options

## 2.54.72 Class Phalcon\Db\Dialect\Mysql

*extends abstract class Phalcon\Db\Dialet*

*implements Phalcon\Db\DialetInterface*

Generates database specific SQL for the MySQL RBDMS

### Methods

**public string getColumnDefinition (Phalcon\Db\ColumnInterface \$column)**

Gets the column name in MySQL

**public string addColumn (string \$tableName, string \$schemaName, Phalcon\Db\ColumnInterface \$column)**

Generates SQL to add a column to a table

**public string modifyColumn (string \$tableName, string \$schemaName, Phalcon\Db\ColumnInterface \$column)**

Generates SQL to modify a column in a table

**public string dropColumn (string \$tableName, string \$schemaName, string \$columnName)**

Generates SQL to delete a column from a table

**public string addIndex (string \$tableName, string \$schemaName, Phalcon\Db\IndexInterface \$index)**

Generates SQL to add an index to a table

**public string dropIndex (string \$tableName, string \$schemaName, string \$indexName)**

Generates SQL to delete an index from a table

**public string addPrimaryKey (string \$tableName, string \$schemaName, Phalcon\Db\IndexInterface \$index)**

Generates SQL to add the primary key to a table

**public string dropPrimaryKey (string \$tableName, string \$schemaName)**

Generates SQL to delete primary key from a table

**public string addForeignKey (string \$tableName, string \$schemaName, Phalcon\Db\ReferenceInterface \$reference)**

Generates SQL to add an index to a table

**public string dropForeignKey (string \$tableName, string \$schemaName, string \$referenceName)**

Generates SQL to delete a foreign key from a table

**protected array \_getTableOptions ()**

Generates SQL to add the table creation options

**public string createTable (string \$tableName, string \$schemaName, array \$definition)**

Generates SQL to create a table in MySQL

**public string dropTable (string \$tableName, string \$schemaName)**

Generates SQL to drop a table

```
public string createView (string $viewName, array $definition, string $schemaName)
```

Generates SQL to create a view

```
public string dropView (string $viewName, string $schemaName, [boolean $IfExists])
```

Generates SQL to drop a view

```
public string tableExists (string $tableName, [string $schemaName])
```

Generates SQL checking for the existence of a schema.table

```
<?php
```

```
echo $dialect->tableExists("posts", "blog");  
echo $dialect->tableExists("posts");
```

```
public string viewExists (string $viewName, [string $schemaName])
```

Generates SQL checking for the existence of a schema.view

```
public string describeColumns (string $table, [string $schema])
```

Generates SQL describing a table

```
<?php
```

```
print_r($dialect->describeColumns("posts")) ?>
```

```
public array listTables ([string $schemaName])
```

List all tables on database

```
<?php
```

```
print_r($dialect->listTables("blog")) ?>
```

```
public array listViews ([string $schemaName])
```

Generates the SQL to list all views of a schema or user

```
public string describeIndexes (string $table, [string $schema])
```

Generates SQL to query indexes on a table

```
public string describeReferences (string $table, [string $schema])
```

Generates SQL to query foreign keys on a table

```
public string tableOptions (string $table, [string $schema])
```

Generates the SQL to describe the table creation options

```
public string limit (string $sqlQuery, int $number) inherited from Phalcon\Db\Dialet
```

Generates the SQL for LIMIT clause

```
<?php
```

```
$sql = $dialect->limit('SELECT * FROM robots', 10);  
echo $sql; // SELECT * FROM robots LIMIT 10
```

```
public string forUpdate (string $sqlQuery) inherited from Phalcon\Db\Dialet
```

Returns a SQL modified with a FOR UPDATE clause

```
<?php
```

```
$sql = $dialect->forUpdate('SELECT * FROM robots');
echo $sql; // SELECT * FROM robots FOR UPDATE
```

**public string sharedLock (string \$sqlQuery)** inherited from Phalcon\Db\Dialect

Returns a SQL modified with a LOCK IN SHARE MODE clause

```
<?php
```

```
$sql = $dialect->sharedLock('SELECT * FROM robots');
echo $sql; // SELECT * FROM robots LOCK IN SHARE MODE
```

**public string getColumnList (array \$columnList)** inherited from Phalcon\Db\Dialect

Gets a list of columns with escaped identifiers

```
<?php
```

```
echo $dialect->getColumnList(array('column1', 'column'));
```

**public string getSqlExpression (array \$expression, [string \$escapeChar])** inherited from Phalcon\Db\Dialect

Transforms an intermediate representation for a expression into a database system valid expression

**public string getSqlTable (array \$table, [string \$escapeChar])** inherited from Phalcon\Db\Dialect

Transform an intermediate representation for a schema/table into a database system valid expression

**public string select (array \$definition)** inherited from Phalcon\Db\Dialect

Builds a SELECT statement

**public boolean supportsSavepoints ()** inherited from Phalcon\Db\Dialect

Checks whether the platform supports savepoints

**public boolean supportsReleaseSavepoints ()** inherited from Phalcon\Db\Dialect

Checks whether the platform supports releasing savepoints.

**public string createSavepoint (string \$name)** inherited from Phalcon\Db\Dialect

Generate SQL to create a new savepoint

**public string releaseSavepoint (string \$name)** inherited from Phalcon\Db\Dialect

Generate SQL to release a savepoint

**public string rollbackSavepoint (string \$name)** inherited from Phalcon\Db\Dialect

Generate SQL to rollback a savepoint

## 2.54.73 Class Phalcon\Db\Dialect\Oracle

*extends abstract class Phalcon\Db\Dialect*

*implements Phalcon\Db\DialectInterface*

Generates database specific SQL for the Oracle RDBMS

## Methods

`public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)`

Gets the column name in Oracle

`public string addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)`

Generates SQL to add a column to a table

`public string modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)`

Generates SQL to modify a column in a table

`public string dropColumn (string $tableName, string $schemaName, string $columnName)`

Generates SQL to delete a column from a table

`public string addIndex (string $tableName, string $schemaName, Phalcon\Db\Index $index)`

Generates SQL to add an index to a table

`public string dropIndex (string $tableName, string $schemaName, string $indexName)`

Generates SQL to delete an index from a table

`public string addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\Index $index)`

Generates SQL to add the primary key to a table

`public string dropPrimaryKey (string $tableName, string $schemaName)`

Generates SQL to delete primary key from a table

`public string addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)`

Generates SQL to add an index to a table

`public string dropForeignKey (string $tableName, string $schemaName, string $referenceName)`

Generates SQL to delete a foreign key from a table

`protected array _getTableOptions ()`

Generates SQL to add the table creation options

`public string createTable (string $tableName, string $schemaName, array $definition)`

Generates SQL to create a table in PostgreSQL

`public boolean dropTable (string $tableName, string $schemaName)`

Generates SQL to drop a table

`public string createView (string $viewName, array $definition, string $schemaName)`

Generates SQL to create a view

`public string dropView (string $viewName, string $schemaName, [boolean $IfExists])`

Generates SQL to drop a view

`public string tableExists (string $tableName, [string $schemaName])`

Generates SQL checking for the existence of a schema.table

```
<?php

var_dump($dialect->tableExists("posts", "blog"));
var_dump($dialect->tableExists("posts"));

public string viewExists (string $viewName, [string $schemaName])
Generates SQL checking for the existence of a schema.view

public string describeColumns (string $table, [string $schema])
Generates a SQL describing a table

<?php

print_r($dialect->describeColumns("posts")); ?>

public array listTables ([string $schemaName])
List all tables on database

<?php

print_r($dialect->listTables("blog")) ?>

public array listViews ([string $schemaName])
Generates the SQL to list all views of a schema or user

public string describeIndexes (string $table, [string $schema])
Generates SQL to query indexes on a table

public string describeReferences (string $table, [string $schema])
Generates SQL to query foreign keys on a table

public string tableOptions (string $table, [string $schema])
Generates the SQL to describe the table creation options

public string getSqlTable (array $table, [string $escapeChar])
Transform an intermediate representation for a schema/table into a database system valid expression

public string limit (string $sqlQuery, int $number)
Generates the SQL for LIMIT clause

<?php

$sql = $dialect->limit('SELECT * FROM robots', 10);
echo $sql; // SELECT * FROM robots LIMIT 10

public string select (array $definition)
Builds a SELECT statement

public boolean supportsSavepoints ()
Checks whether the platform supports savepoints

public boolean supportsReleaseSavepoints ()
Checks whether the platform supports releasing savepoints.

public string forUpdate (string $sqlQuery) inherited from Phalcon\Db\Dialet
```

Returns a SQL modified with a FOR UPDATE clause

```
<?php  
  
$sql = $dialect->forUpdate('SELECT * FROM robots');  
echo $sql; // SELECT * FROM robots FOR UPDATE
```

public *string* **sharedLock** (*string* \$sqlQuery) inherited from Phalcon\Db\Dialet

Returns a SQL modified with a LOCK IN SHARE MODE clause

```
<?php  
  
$sql = $dialect->sharedLock('SELECT * FROM robots');  
echo $sql; // SELECT * FROM robots LOCK IN SHARE MODE
```

public *string* **getColumnList** (*array* \$columnList) inherited from Phalcon\Db\Dialet

Gets a list of columns with escaped identifiers

```
<?php  
  
echo $dialect->getColumnList(array('column1', 'column'));
```

public *string* **getSqlExpression** (*array* \$expression, [*string* \$escapeChar]) inherited from Phalcon\Db\Dialet

Transforms an intermediate representation for a expression into a database system valid expression

public *string* **createSavepoint** (*string* \$name) inherited from Phalcon\Db\Dialet

Generate SQL to create a new savepoint

public *string* **releaseSavepoint** (*string* \$name) inherited from Phalcon\Db\Dialet

Generate SQL to release a savepoint

public *string* **rollbackSavepoint** (*string* \$name) inherited from Phalcon\Db\Dialet

Generate SQL to rollback a savepoint

## 2.54.74 Class Phalcon\Db\Dialet\Postgresql

*extends* abstract class Phalcon\Db\Dialet

*implements* Phalcon\Db\DialetInterface

Generates database specific SQL for the PostgreSQL RDBMS

### Methods

public *string* **getColumnDefinition** (*Phalcon\Db\ColumnInterface* \$column)

Gets the column name in PostgreSQL

public *string* **addColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column)

Generates SQL to add a column to a table

public *string* **modifyColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column)

Generates SQL to modify a column in a table

`public string dropColumn (string $tableName, string $schemaName, string $columnName)`

Generates SQL to delete a column from a table

`public string addIndex (string $tableName, string $schemaName, Phalcon\Db\Index $index)`

Generates SQL to add an index to a table

`public string dropIndex (string $tableName, string $schemaName, string $indexName)`

Generates SQL to delete an index from a table

`public string addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\Index $index)`

Generates SQL to add the primary key to a table

`public string dropPrimaryKey (string $tableName, string $schemaName)`

Generates SQL to delete primary key from a table

`public string addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)`

Generates SQL to add an index to a table

`public string dropForeignKey (string $tableName, string $schemaName, string $referenceName)`

Generates SQL to delete a foreign key from a table

`protected array _getTableOptions ()`

Generates SQL to add the table creation options

`public string createTable (string $tableName, string $schemaName, array $definition)`

Generates SQL to create a table in PostgreSQL

`public boolean dropTable (string $tableName, string $schemaName)`

Generates SQL to drop a table

`public string createView (string $viewName, array $definition, string $schemaName)`

Generates SQL to create a view

`public string dropView (string $viewName, string $schemaName, [boolean $IfExists])`

Generates SQL to drop a view

`public string tableExists (string $tableName, [string $schemaName])`

Generates SQL checking for the existence of a schema.table <code>echo \$dialect->tableExists("posts", "blog")</code>echo \$dialect->tableExists("posts")

`public string viewExists (string $viewName, [string $schemaName])`

Generates SQL checking for the existence of a schema.view

`public string describeColumns (string $table, [string $schema])`

Generates a SQL describing a table <code>print\_r(\$dialect->describeColumns("posts")) ?>

`public array listTables ([string $schemaName])`

List all tables on database

```
<?php

print_r($dialect->listTables("blog")) ?>

public array listViews ([string $schemaName])
Generates the SQL to list all views of a schema or user

public string describeIndexes (string $table, [string $schema])
Generates SQL to query indexes on a table

public string describeReferences (string $table, [string $schema])
Generates SQL to query foreign keys on a table

public string tableOptions (string $table, [string $schema])
Generates the SQL to describe the table creation options

public string limit (string $sqlQuery, int $number) inherited from Phalcon\Db\Dialet
Generates the SQL for LIMIT clause

<?php

$sql = $dialect->limit('SELECT * FROM robots', 10);
echo $sql; // SELECT * FROM robots LIMIT 10

public string forUpdate (string $sqlQuery) inherited from Phalcon\Db\Dialet
Returns a SQL modified with a FOR UPDATE clause

<?php

$sql = $dialect->forUpdate('SELECT * FROM robots');
echo $sql; // SELECT * FROM robots FOR UPDATE

public string sharedLock (string $sqlQuery) inherited from Phalcon\Db\Dialet
Returns a SQL modified with a LOCK IN SHARE MODE clause

<?php

$sql = $dialect->sharedLock('SELECT * FROM robots');
echo $sql; // SELECT * FROM robots LOCK IN SHARE MODE

public string getColumnList (array $columnList) inherited from Phalcon\Db\Dialet
Gets a list of columns with escaped identifiers

<?php

echo $dialect->getColumnList(array('column1', 'column'));

public string getSqlExpression (array $expression, [string $escapeChar]) inherited from
Phalcon\Db\Dialet
Transforms an intermediate representation for a expression into a database system valid expression

public string getSqlTable (array $table, [string $escapeChar]) inherited from Phalcon\Db\Dialet
Transform an intermediate representation for a schema/table into a database system valid expression

public string select (array $definition) inherited from Phalcon\Db\Dialet
```

Builds a SELECT statement

`public boolean supportsSavepoints ()` inherited from Phalcon\Db\Dialect

Checks whether the platform supports savepoints

`public boolean supportsReleaseSavepoints ()` inherited from Phalcon\Db\Dialect

Checks whether the platform supports releasing savepoints.

`public string createSavepoint (string $name)` inherited from Phalcon\Db\Dialect

Generate SQL to create a new savepoint

`public string releaseSavepoint (string $name)` inherited from Phalcon\Db\Dialect

Generate SQL to release a savepoint

`public string rollbackSavepoint (string $name)` inherited from Phalcon\Db\Dialect

Generate SQL to rollback a savepoint

## 2.54.75 Class Phalcon\Db\Dialect\Sqlite

*extends abstract class Phalcon\Db\Dialect*

*implements Phalcon\Db\DialectInterface*

Generates database specific SQL for the SQLite RDBMS

### Methods

`public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)`

Gets the column name in Sqlite

`public string addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)`

Generates SQL to add a column to a table

`public string modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)`

Generates SQL to modify a column in a table

`public string dropColumn (string $tableName, string $schemaName, string $columnName)`

Generates SQL to delete a column from a table

`public string addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)`

Generates SQL to add an index to a table

`public string dropIndex (string $tableName, string $schemaName, string $indexName)`

Generates SQL to delete an index from a table

`public string addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)`

Generates SQL to add the primary key to a table

`public string dropPrimaryKey (string $tableName, string $schemaName)`

Generates SQL to delete primary key from a table

public *string* **addForeignKey** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\Reference* \$reference)  
Generates SQL to add an index to a table

public *string* **dropForeignKey** (*string* \$tableName, *string* \$schemaName, *string* \$referenceName)  
Generates SQL to delete a foreign key from a table

protected *array* **\_getTableOptions** ()  
Generates SQL to add the table creation options

public *string* **createTable** (*string* \$tableName, *string* \$schemaName, *array* \$definition)  
Generates SQL to create a table in Sqlite

public *boolean* **dropTable** (*string* \$tableName, *string* \$schemaName)  
Generates SQL to drop a table

public *string* **createView** (*string* \$viewName, *array* \$definition, *string* \$schemaName)  
Generates SQL to create a view

public *string* **dropView** (*string* \$viewName, *string* \$schemaName, [*boolean* \$IfExists])  
Generates SQL to drop a view

public *string* **tableExists** (*string* \$tableName, [*string* \$schemaName])  
Generates SQL checking for the existence of a schema.table <code>echo \$dialect->tableExists("posts", "blog")</code>

public *string* **viewExists** (*string* \$viewName, [*string* \$schemaName])  
Generates SQL checking for the existence of a schema.view

public *string* **describeColumns** (*string* \$table, [*string* \$schema])  
Generates a SQL describing a table <code>print\_r(\$dialect->describeColumns("posts")) ?>

public *array* **listTables** ([*string* \$schemaName])  
List all tables on database <code>print\_r(\$dialect->listTables("blog")) ?>

public *array* **listViews** ([*string* \$schemaName])  
Generates the SQL to list all views of a schema or user

public *string* **describeIndexes** (*string* \$table, [*string* \$schema])  
Generates SQL to query indexes on a table

public *string* **describeIndex** (*string* \$indexName)  
Generates SQL to query indexes detail on a table

public *string* **describeReferences** (*string* \$table, [*string* \$schema])  
Generates SQL to query foreign keys on a table

public *string* **tableOptions** (*string* \$table, [*string* \$schema])  
Generates the SQL to describe the table creation options

public *string* **limit** (*string* \$sqlQuery, *int* \$number) inherited from Phalcon\Db\Dialet  
Generates the SQL for LIMIT clause

```
<?php
```

```
$sql = $dialect->limit('SELECT * FROM robots', 10);
echo $sql; // SELECT * FROM robots LIMIT 10
```

*public string forUpdate (string \$sqlQuery)* inherited from Phalcon\Db\Dialect

Returns a SQL modified with a FOR UPDATE clause

```
<?php
```

```
$sql = $dialect->forUpdate('SELECT * FROM robots');
echo $sql; // SELECT * FROM robots FOR UPDATE
```

*public string sharedLock (string \$sqlQuery)* inherited from Phalcon\Db\Dialect

Returns a SQL modified with a LOCK IN SHARE MODE clause

```
<?php
```

```
$sql = $dialect->sharedLock('SELECT * FROM robots');
echo $sql; // SELECT * FROM robots LOCK IN SHARE MODE
```

*public string getColumnList (array \$columnList)* inherited from Phalcon\Db\Dialect

Gets a list of columns with escaped identifiers

```
<?php
```

```
echo $dialect->getColumnList(array('column1', 'column'));
```

*public string getSqlExpression (array \$expression, [string \$escapeChar])* inherited from Phalcon\Db\Dialect

Transforms an intermediate representation for a expression into a database system valid expression

*public string getSqlTable (array \$table, [string \$escapeChar])* inherited from Phalcon\Db\Dialect

Transform an intermediate representation for a schema/table into a database system valid expression

*public string select (array \$definition)* inherited from Phalcon\Db\Dialect

Builds a SELECT statement

*public boolean supportsSavepoints ()* inherited from Phalcon\Db\Dialect

Checks whether the platform supports savepoints

*public boolean supportsReleaseSavepoints ()* inherited from Phalcon\Db\Dialect

Checks whether the platform supports releasing savepoints.

*public string createSavepoint (string \$name)* inherited from Phalcon\Db\Dialect

Generate SQL to create a new savepoint

*public string releaseSavepoint (string \$name)* inherited from Phalcon\Db\Dialect

Generate SQL to release a savepoint

*public string rollbackSavepoint (string \$name)* inherited from Phalcon\Db\Dialect

Generate SQL to rollback a savepoint

## 2.54.76 Class Phalcon\Db\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Db will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.77 Class Phalcon\Db\Index

*implements Phalcon\Db\IndexInterface*

Allows to define indexes to be used on tables. Indexes are a common way to enhance database performance. An index allows the database server to find and retrieve specific rows much faster than it could do without an index

### Methods

public **\_\_construct** (*string \$indexName*, *array \$columns*)

Phalcon\Db\Index constructor

public *string getName* ()

Gets the index name

`public array getColumns ()`

Gets the columns that comprehends the index

`public static Phalcon\Db\IndexInterface __set_state ([unknown $properties])`

Restore a Phalcon\Db\Index object from export

## 2.54.78 Class Phalcon\Db\Profiler

Instances of Phalcon\Db can generate execution profiles on SQL statements sent to the relational database. Profiled information includes execution time in milliseconds. This helps you to identify bottlenecks in your applications.

```
<?php

$profiler = new Phalcon\Db\Profiler();

//Set the connection profiler
$connection->setProfiler($profiler);

$sql = "SELECT buyer_name, quantity, product_name
FROM buyers LEFT JOIN products ON
buyers.pid=products.id";

//Execute a SQL statement
$connection->query($sql);

//Get the last profile in the profiler
$profile = $profiler->getLastProfile();

echo "SQL Statement: ", $profile->getSQLStatement(), "\n";
echo "Start Time: ", $profile->getInitialTime(), "\n";
echo "Final Time: ", $profile->getFinalTime(), "\n";
echo "Total Elapsed Time: ", $profile->getTotalElapsedSeconds(), "\n";
```

### Methods

`public Phalcon\Db\Profiler startProfile (string $sqlStatement, [unknown $sqlVariables], [unknown $sqlBindTypes])`

Starts the profile of a SQL sentence

`public Phalcon\Db\Profiler stopProfile ()`

Stops the active profile

`public integer getNumberTotalStatements ()`

Returns the total number of SQL statements processed

`public double getTotalElapsedSeconds ()`

Returns the total time in seconds spent by the profiles

`public Phalcon\Db\Profiler\Item [] getProfiles ()`

Returns all the processed profiles

public *Phalcon\Db\Profiler* **reset** ()

Resets the profiler, cleaning up all the profiles

public *Phalcon\Db\Profiler\Item* **getLastProfile** ()

Returns the last profile executed in the profiler

## 2.54.79 Class Phalcon\Db\Profiler\Item

This class identifies each profile in a Phalcon\Db\Profiler

### Methods

public **setSQLStatement** (*string* \$sqlStatement)

Sets the SQL statement related to the profile

public *string* **getSQLStatement** ()

Returns the SQL statement related to the profile

public **setSQLVariables** (*unknown* \$sqlVariables)

Sets the SQL variables related to the profile

public *array* **getSQLVariables** ()

Returns the SQL variables related to the profile

public **setSQLBindTypes** (*unknown* \$sqlBindTypes)

Sets the SQL bind types related to the profile

public *array* **getSQLBindTypes** ()

Returns the SQL bind types related to the profile

public **setInitialTime** (*int* \$initialTime)

Sets the timestamp on when the profile started

public **setFinalTime** (*int* \$finalTime)

Sets the timestamp on when the profile ended

public *double* **getInitialTime** ()

Returns the initial time in miliseconds on when the profile started

public *double* **getFinalTime** ()

Returns the initial time in miliseconds on when the profile ended

public *double* **getTotalElapsedSeconds** ()

Returns the total time in seconds spent by the profile

## 2.54.80 Class Phalcon\Db\RawValue

This class allows to insert/update raw data without quoting or formating. The next example shows how to use the MySQL now() function as a field value.

```
<?php

$subscriber = new Subscribers();
$subscriber->email = 'andres@phalconphp.com';
$subscriber->created_at = new Phalcon\Db\RawValue('now()');
$subscriber->save();
```

## Methods

public **\_\_construct** (*string* \$value)

Phalcon\Db\RawValue constructor

public **getValue** ()

Returns internal raw value without quoting or formating

public **\_\_toString** ()

Magic method \_\_toString returns raw value without quoting or formating

## 2.54.81 Class Phalcon\Db\Reference

*implements Phalcon\Db\ReferenceInterface*

Allows to define reference constraints on tables

```
<?php
```

```
$reference = new Phalcon\Db\Reference("field_fk", array(
    'referencedSchema' => "invoicing",
    'referencedTable' => "products",
    'columns' => array("product_type", "product_code"),
    'referencedColumns' => array("type", "code")
));
```

## Methods

public **\_\_construct** (*string* \$referenceName, *array* \$definition)

Phalcon\Db\Reference constructor

public **getName** ()

Gets the index name

public **getSchemaName** ()

Gets the schema where referenced table is

public **getReferencedSchema** ()

Gets the schema where referenced table is

public **getColumns** ()

Gets local columns which reference is based

public **getReferencedTable** ()

Gets the referenced table

```
public array getReferencedColumns ()  
Gets referenced columns  
public static Phalcon\Db\Reference __set_state ([unknown $properties])  
Restore a Phalcon\Db\Reference object from export
```

## 2.54.82 Class Phalcon\Db\Result\Pdo

Encapsulates the resultset internals

```
<?php
```

```
$result = $connection->query("SELECT * FROM robots ORDER BY name");  
$result->setFetchMode(Phalcon\Db::FETCH_NUM);  
while ($robot = $result->fetchArray()) {  
    print_r($robot);  
}
```

### Methods

```
public __construct (Phalcon\Db\AdapterInterface $connection, PDOStatement $result, [string $sqlStatement], [array $bindParams], [array $bindTypes])
```

Phalcon\Db\Result\Pdo constructor

```
public boolean execute ()
```

Allows to executes the statement again. Some database systems don't support scrollable cursors, So, as cursors are forward only, we need to execute the cursor again to fetch rows from the begining

```
public mixed fetch ()
```

Fetches an array/object of strings that corresponds to the fetched row, or FALSE if there are no more rows. This method is affected by the active fetch flag set using Phalcon\Db\Result\Pdo::setFetchMode

```
<?php
```

```
$result = $connection->query("SELECT * FROM robots ORDER BY name");  
$result->setFetchMode(Phalcon\Db::FETCH_OBJ);  
while ($robot = $result->fetch()) {  
    echo $robot->name;  
}
```

```
public mixed fetchArray ()
```

Returns an array of strings that corresponds to the fetched row, or FALSE if there are no more rows. This method is affected by the active fetch flag set using Phalcon\Db\Result\Pdo::setFetchMode

```
<?php
```

```
$result = $connection->query("SELECT * FROM robots ORDER BY name");  
$result->setFetchMode(Phalcon\Db::FETCH_NUM);  
while ($robot = $result->fetchArray()) {  
    print_r($robot);  
}
```

**public array fetchAll ()**

Returns an array of arrays containing all the records in the result This method is affected by the active fetch flag set using Phalcon\Db\Result\Pdo::setFetchMode

<?php

```
$result = $connection->query("SELECT * FROM robots ORDER BY name");
$robots = $result->fetchAll();
```

**public int numRows ()**

Gets number of rows returned by a resulset

<?php

```
$result = $connection->query("SELECT * FROM robots ORDER BY name");
echo 'There are ', $result->numRows(), ' rows in the resulset';
```

**public dataSeek (int \$number)**

Moves internal resulset cursor to another position letting us to fetch a certain row

<?php

```
$result = $connection->query("SELECT * FROM robots ORDER BY name");
$result->dataSeek(2); // Move to third row on result
$row = $result->fetch(); // Fetch third row
```

**public setFetchMode (int \$fetchMode, [unknown \$fetchArg1], [unknown \$fetchArg2])**

Changes the fetching mode affecting Phalcon\Db\Result\Pdo::fetch()

<?php

```
//Return array with integer indexes
$result->setFetchMode(Phalcon\Db::FETCH_NUM);
```

```
//Return associative array without integer indexes
$result->setFetchMode(Phalcon\Db::FETCH_ASSOC);
```

```
//Return associative array together with integer indexes
$result->setFetchMode(Phalcon\Db::FETCH_BOTH);
```

```
//Return an object
$result->setFetchMode(Phalcon\Db::FETCH_OBJ);
```

**public PDOStatement getInternalResult ()**

Gets the internal PDO result object

**public boolean nextRowset ()**

Advances to the next rowset in a multi-rowset statement handle

## 2.54.83 Class Phalcon\Debug

Provides debug capabilities to Phalcon applications

## Methods

public *Phalcon\Debug* **setUri** (*string* \$uri)

Change the base URI for static resources

public *Phalcon\Debug* **setShowBackTrace** (*boolean* \$showBackTrace)

Sets if files the exception's backtrace must be showed

public *Phalcon\Debug* **setShowFiles** (*boolean* \$showFiles)

Set if files part of the backtrace must be shown in the output

public *Phalcon\Debug* **setShowFileFragment** (*boolean* \$showFileFragment)

Sets if files must be completely opened and showed in the output or just the fragment related to the exception

public *Phalcon\Debug* **listen** ([*boolean* \$exceptions], [*boolean* \$lowSeverity])

Listen for uncaught exceptions and unsilent notices or warnings

public *Phalcon\Debug* **listenExceptions** ()

Listen for uncaught exceptions

public *Phalcon\Debug* **listenLowSeverity** ()

Listen for unsilent notices or warnings

public *Phalcon\Debug* **debugVar** (*mixed* \$var, [*string* \$key])

Adds a variable to the debug output

public *Phalcon\Debug* **clearVars** ()

Clears are variables added previously

protected *string* **\_escapeString** ()

Escapes a string with htmlentities

protected *string* **\_getArrayDump** ()

Produces a recursive representation of an array

protected *string* **\_getVarDump** ()

Produces an string representation of a variable

public *string* **getMajorVersion** ()

Returns the major framework's version

public *string* **getVersion** ()

Generates a link to the current version documentation

public *string* **getCssSources** ()

Returns the css sources

public *string* **getJsSources** ()

Returns the javascript sources

protected **showTraceItem** ()

Shows a backtrace item

public *boolean* **onUncaughtException** (*Exception* \$exception)

Handles uncaught exceptions

**public string getCharset ()**

Returns the character set used to display the HTML

**public Phalcon\Debug setCharset (string \$charset)**

Sets the character set used to display the HTML

**public int getLinesBeforeContext ()**

Returns the number of lines displayed before the error line

**public Phalcon\Debug setLinesBeforeContext (int \$lines)**

Sets the number of lines displayed before the error line

**public int getLinesAfterContext ()**

Returns the number of lines displayed after the error line

**public Phalcon\Debug setLinesAfterContext (int \$lines)**

Sets the number of lines displayed after the error line

**protected getFileLink (unknown \$file, unknown \$line, unknown \$format)**

...

## 2.54.84 Abstract class Phalcon\Dispatcher

*implements* [Phalcon\DispatcherInterface](#), [Phalcon\DI\InjectionAwareInterface](#),  
[Phalcon\Events\EventsAwareInterface](#)

This is the base class for Phalcon\Mvc\Dispatcher and Phalcon\CLI\Dispatcher. This class can't be instantiated directly, you can use it to create your own dispatchers

### Constants

*integer EXCEPTION\_NO\_DI*

*integer EXCEPTION\_CYCLIC\_ROUTING*

*integer EXCEPTION\_HANDLER\_NOT\_FOUND*

*integer EXCEPTION\_INVALID\_HANDLER*

*integer EXCEPTION\_INVALID\_PARAMS*

*integer EXCEPTION\_ACTION\_NOT\_FOUND*

### Methods

**public \_\_construct ()**

Phalcon\Dispatcher constructor

**public setDI ([Phalcon\DiInterface](#) \$dependencyInjector)**

Sets the dependency injector

**public [Phalcon\DiInterface](#) getDI ()**

Returns the internal dependency injector

**public setEventsManager (*Phalcon\Events\ManagerInterface* \$eventsManager)**

Sets the events manager

**public *Phalcon\Events\ManagerInterface* getEventsManager ()**

Returns the internal event manager

**public setActionSuffix (*string* \$actionSuffix)**

Sets the default action suffix

**public setModuleName (*string* \$moduleName)**

Sets the module where the controller is (only informative)

**public *string* getModuleName ()**

Gets the module where the controller class is

**public setNamespaceName (*string* \$namespaceName)**

Sets the namespace where the controller class is

**public *string* getNamespaceName ()**

Gets a namespace to be prepended to the current handler name

**public setDefaultNamespace (*string* \$namespace)**

Sets the default namespace

**public *string* getDefaultNamespace ()**

Returns the default namespace

**public setDefaultAction (*string* \$actionName)**

Sets the default action name

**public setActionName (*string* \$actionName)**

Sets the action name to be dispatched

**public *string* getActionName ()**

Gets the lastest dispatched action name

**public setParams (*array* \$params)**

Sets action params to be dispatched

**public *array* getParams ()**

Gets action params

**public setParam (*mixed* \$param, *mixed* \$value)**

Set a param by its name or numeric index

**public *mixed* getParam (*mixed* \$param, [*string/array* \$filters])**

Gets a param by its name or numeric index

**public *string* getActiveMethod ()**

Returns the current method to be/executed in the dispatcher

**public *boolean* isFinished ()**

Checks if the dispatch loop is finished or has more pendent controllers/tasks to disptach

**public `setReturnedValue` (*mixed* \$value)**

Sets the latest returned value by an action manually

**public *mixed* `getReturnedValue` ()**

Returns value returned by the lastest dispatched action

**public *object* `dispatch` ()**

Dispatches a handle action taking into account the routing parameters

**public `forward` (*array* \$forward)**

Forwards the execution flow to another controller/action Dispatchers are unique per module. Forwarding between modules is not allowed

<?php

```
$this->dispatcher->forward(array('controller' => 'posts', 'action' => 'index'));
```

**public *boolean* `wasForwarded` ()**

Check if the current executed action was forwarded by another one

**public *string* `getHandlerClass` ()**

Possible class name that will be located to dispatch the request

## 2.54.85 Class Phalcon\Escaper

*implements Phalcon\EscaperInterface*

Escapes different kinds of text securing them. By using this component you may prevent XSS attacks. This component only works with UTF-8. The PREG extension needs to be compiled with UTF-8 support.

<?php

```
$escaper = new Phalcon\Escaper();
$escaped = $escaper->escapeCss("font-family: <Verdana>");
echo $escaped; // font\2D family\3A \20 \3C Verdana\3E
```

### Methods

**public `setEncoding` (*string* \$encoding)**

Sets the encoding to be used by the escaper

<?php

```
$escaper->setEncoding('utf-8');
```

**public *string* `getEncoding` ()**

Returns the internal encoding used by the escaper

**public `setHtmlQuoteType` (*int* \$quoteType)**

Sets the HTML quoting type for htmlspecialchars

```
<?php  
  
$escaper->setHtmlQuoteType(ENT_XHTML);  
  
public string detectEncoding (string $str)  
Detect the character encoding of a string to be handled by an encoder Special-handling for chr(172) and  
chr(128) to chr(159) which fail to be detected by mb_detect_encoding()  
  
public string normalizeEncoding (string $str)  
Utility to normalize a string's encoding to UTF-32.  
  
public string escapeHtml (string $text)  
Escapes a HTML string. Internally uses htmlspecialchars  
  
public string escapeHtmlAttr (unknown $text)  
Escapes a HTML attribute string  
  
public string escapeCss (string $css)  
Escape CSS strings by replacing non-alphanumeric chars by their hexadecimal escaped representation  
  
public string escapeJs (string $js)  
Escape javascript strings by replacing non-alphanumeric chars by their hexadecimal escaped representation  
  
public string escapeUrl (string $url)  
Escapes a URL. Internally uses rawurlencode
```

## 2.54.86 Class Phalcon\Escaper\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Escaper will use this class

### Methods

final private *Exception* \_\_clone () inherited from Exception

Clone the exception

public \_\_construct ([string \$message], [int \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public string getMessage () inherited from Exception

Gets the Exception message

final public int getCode () inherited from Exception

Gets the Exception code

final public string getFile () inherited from Exception

Gets the file in which the exception occurred

final public int getLine () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception  
Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception  
Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception  
Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception  
String representation of the exception

## 2.54.87 Class Phalcon\Events\Event

This class offers contextual information of a fired event in the EventsManager

### Methods

public **\_\_construct** (*string* \$type, *object* \$source, [*mixed* \$data], [*boolean* \$cancelable])  
Phalcon\Events\Event constructor

public **setType** (*string* \$eventType)  
Set the event's type

public *string* **getType** ()  
Returns the event's type

public *object* **getSource** ()  
Returns the event's source

public *string* **setData** (*string* \$data)  
Set the event's data

public *mixed* **getData** ()  
Returns the event's data

public *boolean* **setCancelable** (*boolean* \$cancelable)  
Sets if the event is cancelable

public *boolean* **getCancelable** ()  
Check whether the event is cancelable

public **stop** ()  
Stops the event preventing propagation

public **isStopped** ()  
Check whether the event is currently stopped

## 2.54.88 Class Phalcon\Events\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Events will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.89 Class Phalcon\Events\Manager

*implements Phalcon\Events\ManagerInterface*

Phalcon Events Manager, offers an easy way to intercept and manipulate, if needed, the normal flow of operation. With the EventsManager the developer can create hooks or plugins that will offer monitoring of data, manipulation, conditional execution and much more.

### Methods

public **attach** (*string \$eventType, object/callable \$handler*)

Attach a listener to the events manager

public **enablePriorities** (*boolean \$enablePriorities*)

Set if priorities are enabled in the EventsManager

**public boolean arePrioritiesEnabled ()**

Returns if priorities are enabled

**public collectResponses (boolean \$collect)**

Tells the event manager if it needs to collect all the responses returned by every registered listener in a single fire

**public isCollecting ()**

Check if the events manager is collecting all all the responses returned by every registered listener in a single fire

**public array getResponses ()**

Returns all the responses returned by every handler executed by the last ‘fire’ executed

**public detachAll ([string \$type])**

Removes all events from the EventsManager

**public mixed fireQueue (SplPriorityQueue \$queue, Phalcon\|Events\|Event \$event)**

Internal handler to call a queue of events

**public mixed fire (string \$eventType, object \$source, [mixed \$data])**

Fires an event in the events manager causing that active listeners be notified about it

<?php

```
$eventsManager->fire('db', $connection);
```

**public boolean hasListeners (string \$type)**

Check whether certain type of event has listeners

**public array getListeners (string \$type)**

Returns all the attached listeners of a certain type

**public dettachAll ([unknown \$type])**

...

## 2.54.90 Class Phalcon\Exception

*extends* Exception

All framework exceptions should use or extend this exception

### Methods

**final private Exception \_\_clone ()** inherited from Exception

Clone the exception

**public \_\_construct ([string \$message], [int \$code], [Exception \$previous])** inherited from Exception

Exception constructor

**final public string getMessage ()** inherited from Exception

Gets the Exception message

`final public int getCode ()` inherited from `Exception`

Gets the Exception code

`final public string getFile ()` inherited from `Exception`

Gets the file in which the exception occurred

`final public int getLine ()` inherited from `Exception`

Gets the line in which the exception occurred

`final public array getTrace ()` inherited from `Exception`

Gets the stack trace

`final public Exception getPrevious ()` inherited from `Exception`

Returns previous Exception

`final public Exception getTraceAsString ()` inherited from `Exception`

Gets the stack trace as a string

`public string __toString ()` inherited from `Exception`

String representation of the exception

## 2.54.91 Class Phalcon\Filter

*implements Phalcon\FilterInterface*

The `Phalcon\Filter` component provides a set of commonly needed data filters. It provides object oriented wrappers to the php filter extension. Also allows the developer to define his/her own filters

<?php

```
$filter = new Phalcon\Filter();
$filter->sanitize("some(one)@exa\\mple.com", "email"); // returns "someone@example.com"
$filter->sanitize("hello<<", "string"); // returns "hello"
$filter->sanitize("!100a019", "int"); // returns "100019"
$filter->sanitize("!100a019.01a", "float"); // returns "100019.01"
```

### Methods

`public Phalcon\Filter add (string $name, callable $handler)`

Adds a user-defined filter

`public mixed sanitize (mixed $value, mixed $filters)`

Sanitizes a value with a specified single or set of filters

`protected mixed __sanitize ()`

Internal sanitize wrapper to `filter_var`

`public object// getFilters ()`

Return the user-defined filters in the instance

## 2.54.92 Class Phalcon\Filter\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Filter will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.93 Abstract class Phalcon\Flash

Shows HTML notifications related to different circumstances. Classes can be stylized using CSS

<?php

```
$flash->success("The record was successfully deleted");
$flash->error("Cannot open the file");
```

## Methods

public **\_\_construct** ([array \$cssClasses])

Phalcon\Flash constructor

public *Phalcon\FlashInterface* **setImplicitFlush** (boolean \$implicitFlush)

Set whether the output must be implicitly flushed to the output or returned as string

public *Phalcon\FlashInterface* **setAutomaticHtml** (boolean \$automaticHtml)

Set if the output must be implicitly formatted with HTML

public *Phalcon\FlashInterface* **setCssClasses** (array \$cssClasses)

Set an array with CSS classes to format the messages

public *string* **error** (*string* \$message)

Shows a HTML error message

<?php

```
$flash->error('This is an error');
```

public *string* **notice** (*string* \$message)

Shows a HTML notice/information message

<?php

```
$flash->notice('This is an information');
```

public *string* **success** (*string* \$message)

Shows a HTML success message

<?php

```
$flash->success('The process was finished successfully');
```

public *string* **warning** (*string* \$message)

Shows a HTML warning message

<?php

```
$flash->warning('Hey, this is important');
```

public **outputMessage** (*string* \$type, *string* \$message)

Outputs a message formatting it with HTML

<?php

```
$flash->outputMessage('error', $message);
```

## 2.54.94 Class Phalcon\Flash\Direct

*extends* abstract class *Phalcon\Flash*

*implements* *Phalcon\FlashInterface*

This is a variant of the Phalcon\Flash that immediately outputs any message passed to it

## Methods

`public string message (string $type, string $message)`

Outputs a message

`public __construct ([array $cssClasses])` inherited from Phalcon\Flash

Phalcon\Flash constructor

`public Phalcon\FlashInterface setImplicitFlush (boolean $implicitFlush)` inherited from Phalcon\Flash

Set whether the output must be implicitly flushed to the output or returned as string

`public Phalcon\FlashInterface setAutomaticHtml (boolean $automaticHtml)` inherited from Phalcon\Flash

Set if the output must be implicitly formatted with HTML

`public Phalcon\FlashInterface setCssClasses (array $cssClasses)` inherited from Phalcon\Flash

Set an array with CSS classes to format the messages

`public string error (string $message)` inherited from Phalcon\Flash

Shows a HTML error message

`<?php`

```
$flash->error('This is an error');
```

`public string notice (string $message)` inherited from Phalcon\Flash

Shows a HTML notice/information message

`<?php`

```
$flash->notice('This is an information');
```

`public string success (string $message)` inherited from Phalcon\Flash

Shows a HTML success message

`<?php`

```
$flash->success('The process was finished successfully');
```

`public string warning (string $message)` inherited from Phalcon\Flash

Shows a HTML warning message

`<?php`

```
$flash->warning('Hey, this is important');
```

`public outputMessage (string $type, string $message)` inherited from Phalcon\Flash

Outputs a message formatting it with HTML

`<?php`

```
$flash->outputMessage('error', $message);
```

## 2.54.95 Class Phalcon\Flash\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Flash will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.96 Class Phalcon\Flash\Session

*extends abstract class Phalcon\Flash*

*implements Phalcon\FlashInterface, Phalcon\DI\InjectionAwareInterface*

Temporarily stores the messages in session, then messages can be printed in the next request

### Methods

public **setDI** (*Phalcon\DiInterface \$dependencyInjector*)

Sets the dependency injector

public *Phalcon\DiInterface getDI* ()

Returns the internal dependency injector

**protected array \_getSessionMessages ()**

Returns the messages stored in session

**protected \_setSessionMessages ()**

Stores the messages in session

**public message (string \$type, string \$message)**

Adds a message to the session flasher

**public array getMessages ([string \$type], [boolean \$remove])**

Returns the messages in the session flasher

**public output ([boolean \$remove])**

Prints the messages in the session flasher

**public has (unknown \$type)**

**bool \Phalcon\Flash\Session::has(string \$type)**

**public \_\_construct ([array \$cssClasses])** inherited from Phalcon\Flash

Phalcon\Flash constructor

**public Phalcon\FlashInterface setImplicitFlush (boolean \$implicitFlush)** inherited from Phalcon\Flash

Set whether the output must be implicitly flushed to the output or returned as string

**public Phalcon\FlashInterface setAutomaticHtml (boolean \$automaticHtml)** inherited from Phalcon\Flash

Set if the output must be implicitly formatted with HTML

**public Phalcon\FlashInterface setCssClasses (array \$cssClasses)** inherited from Phalcon\Flash

Set an array with CSS classes to format the messages

**public string error (string \$message)** inherited from Phalcon\Flash

Shows a HTML error message

<?php

```
$flash->error('This is an error');
```

**public string notice (string \$message)** inherited from Phalcon\Flash

Shows a HTML notice/information message

<?php

```
$flash->notice('This is an information');
```

**public string success (string \$message)** inherited from Phalcon\Flash

Shows a HTML success message

<?php

```
$flash->success('The process was finished successfully');
```

public *string* **warning** (*string* \$message) inherited from Phalcon\Flash

Shows a HTML warning message

<?php

```
$flash->warning('Hey, this is important');
```

public **outputMessage** (*string* \$type, *string* \$message) inherited from Phalcon\Flash

Outputs a message formatting it with HTML

<?php

```
$flash->outputMessage('error', $message);
```

## 2.54.97 Abstract class Phalcon\Forms\Element

*implements Phalcon\Forms\ElementInterface*

This is a base class for form elements

### Methods

public **\_\_construct** (*string* \$name, [*array* \$attributes])

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form)

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface* **getForm** ()

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name)

Sets the element's name

public *string* **getName** ()

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters)

Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter)

Adds a filter to current list of filters

public *mixed* **getFilters** ()

Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge])

Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator)

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** ()

Returns the validators registered for the element

`public array prepareAttributes ([array $attributes], [boolean $useChecked])`

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

`public Phalcon\Forms\ElementInterface setAttribute (string $attribute, mixed $value)`

Sets a default attribute for the element

`public mixed getAttribute (string $attribute, [mixed $defaultValue])`

Returns the value of an attribute if present

`public Phalcon\Forms\ElementInterface setAttributes (array $attributes)`

Sets default attributes for the element

`public array getAttributes ()`

Returns the default attributes for the element

`public Phalcon\Forms\ElementInterface setUserOption (string $option, mixed $value)`

Sets an option for the element

`public mixed getUserOption (string $option, [mixed $defaultValue])`

Returns the value of an option if present

`public Phalcon\Forms\ElementInterface setUserOptions (array $options)`

Sets options for the element

`public array getUserOptions ()`

Returns the options for the element

`public Phalcon\Forms\ElementInterface setLabel (string $label)`

Sets the element label

`public string getLabel ()`

Returns the element's label

`public string label (unknown $attributes)`

Generate the HTML to label the element

`public Phalcon\Forms\ElementInterface setDefault (mixed $value)`

Sets a default value in case the form does not use an entity or there is no value available for the element in `$_POST`

`public mixed getDefault ()`

Returns the default value assigned to the element

`public mixed getValue ()`

Returns the element's value

`public Phalcon\Validation\Message\Group getMessages ()`

Returns the messages that belongs to the element The element needs to be attached to a form

`public boolean hasMessages ()`

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group)

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message)

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** ()

Clears every element in the form to its default value

public *string* **\_\_toString** ()

Magic method `__toString` renders the widget without attributes

abstract public *string* **render** ([*array* \$attributes]) inherited from *Phalcon\Forms\ElementInterface*

Renders the element widget

## 2.54.98 Class Phalcon\Forms\Element\Check

*extends abstract class Phalcon\Forms\Element*

*implements Phalcon\Forms\ElementInterface*

Component INPUT[type=check] for forms

### Methods

public *string* **render** ([*array* \$attributes])

Renders the element widget returning html

public **\_\_construct** (*string* \$name, [*array* \$attributes]) inherited from *Phalcon\Forms\Element*

*Phalcon\Forms\Element* constructor

public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form) inherited from *Phalcon\Forms\Element*

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface* **getForm** () inherited from *Phalcon\Forms\Element*

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name) inherited from *Phalcon\Forms\Element*

Sets the element's name

public *string* **getName** () inherited from *Phalcon\Forms\Element*

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters) inherited from *Phalcon\Forms\Element*

Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter) inherited from *Phalcon\Forms\Element*

Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from *Phalcon\Forms\Element*

Returns the element's filters

```
public Phalcon\Forms\ElementInterface addValidators (unknown $validators, [unknown $merge])  
inherited from Phalcon\Forms\Element
```

Adds a group of validators

```
public Phalcon\Forms\ElementInterface addValidator (unknown $validator) inherited from  
Phalcon\Forms\Element
```

Adds a validator to the element

```
public Phalcon\Validation\ValidatorInterface [] getValidators () inherited from Phalcon\Forms\Element
```

Returns the validators registered for the element

```
public array prepareAttributes ([array $attributes], [boolean $useChecked]) inherited from  
Phalcon\Forms\Element
```

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

```
public Phalcon\Forms\ElementInterface setAttribute (string $attribute, mixed $value) inherited from  
Phalcon\Forms\Element
```

Sets a default attribute for the element

```
public mixed getAttribute (string $attribute, [mixed $defaultValue]) inherited from  
Phalcon\Forms\Element
```

Returns the value of an attribute if present

```
public Phalcon\Forms\ElementInterface setAttributes (array $attributes) inherited from  
Phalcon\Forms\Element
```

Sets default attributes for the element

```
public array getAttributes () inherited from Phalcon\Forms\Element
```

Returns the default attributes for the element

```
public Phalcon\Forms\ElementInterface setUserOption (string $option, mixed $value) inherited from  
Phalcon\Forms\Element
```

Sets an option for the element

```
public mixed getUserOption (string $option, [mixed $defaultValue]) inherited from  
Phalcon\Forms\Element
```

Returns the value of an option if present

```
public Phalcon\Forms\ElementInterface setUserOptions (array $options) inherited from  
Phalcon\Forms\Element
```

Sets options for the element

```
public array getUserOptions () inherited from Phalcon\Forms\Element
```

Returns the options for the element

```
public Phalcon\Forms\ElementInterface setLabel (string $label) inherited from Phalcon\Forms\Element
```

Sets the element label

```
public string getLabel () inherited from Phalcon\Forms\Element
```

Returns the element's label

```
public string label (unknown $attributes) inherited from Phalcon\Forms\Element
```

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed \$value*) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed getDefault* () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed getValue* () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group getMessages* () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean hasMessages* () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface setMessages* (*Phalcon\Validation\Message\Group \$group*)  
inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface appendMessage* (*Phalcon\Validation\Message \$message*)  
inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element clear* () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string \_\_toString* () inherited from Phalcon\Forms\Element

Magic method *\_\_toString* renders the widget without attributes

## 2.54.99 Class Phalcon\Forms\Element\Date

*extends abstract class Phalcon\Forms\Element*

*implements Phalcon\Forms\ElementInterface*

Component INPUT[type=date] for forms

### Methods

public *string render* ([*array \$attributes*])

Renders the element widget returning html

public *\_\_construct* (*string \$name, [array \$attributes]*) inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface setForm* (*Phalcon\Forms\Form \$form*) inherited from  
Phalcon\Forms\Element

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface getForm* () inherited from Phalcon\Forms\Element

Returns the parent form to the element

`public Phalcon\Forms\ElementInterface setName (string $name)` inherited from `Phalcon\Forms\Element`

Sets the element's name

`public string getName ()` inherited from `Phalcon\Forms\Element`

Returns the element's name

`public Phalcon\Forms\ElementInterface setFilters (array/string $filters)` inherited from `Phalcon\Forms\Element`

Sets the element's filters

`public Phalcon\Forms\ElementInterface addFilter (string $filter)` inherited from `Phalcon\Forms\Element`

Adds a filter to current list of filters

`public mixed getFilters ()` inherited from `Phalcon\Forms\Element`

Returns the element's filters

`public Phalcon\Forms\ElementInterface addValidators (unknown $validators, [unknown $merge])` inherited from `Phalcon\Forms\Element`

Adds a group of validators

`public Phalcon\Forms\ElementInterface addValidator (unknown $validator)` inherited from `Phalcon\Forms\Element`

Adds a validator to the element

`public Phalcon\Validation\ValidatorInterface [] getValidators ()` inherited from `Phalcon\Forms\Element`

Returns the validators registered for the element

`public array prepareAttributes ([array $attributes], [boolean $useChecked])` inherited from `Phalcon\Forms\Element`

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

`public Phalcon\Forms\ElementInterface setAttribute (string $attribute, mixed $value)` inherited from `Phalcon\Forms\Element`

Sets a default attribute for the element

`public mixed getAttribute (string $attribute, [mixed $defaultValue])` inherited from `Phalcon\Forms\Element`

Returns the value of an attribute if present

`public Phalcon\Forms\ElementInterface setAttributes (array $attributes)` inherited from `Phalcon\Forms\Element`

Sets default attributes for the element

`public array getAttributes ()` inherited from `Phalcon\Forms\Element`

Returns the default attributes for the element

`public Phalcon\Forms\ElementInterface setUserOption (string $option, mixed $value)` inherited from `Phalcon\Forms\Element`

Sets an option for the element

`public mixed getUserOption (string $option, [mixed $defaultValue])` inherited from `Phalcon\Forms\Element`

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group) inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message) inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element

Magic method **\_\_toString** renders the widget without attributes

## 2.54.100 Class Phalcon\Forms\Element\Email

*extends* abstract class *Phalcon\Forms\Element*

*implements* *Phalcon\Forms\ElementInterface*

Component INPUT[type=email] for forms

## Methods

`public string render ([array $attributes])`

Renders the element widget returning html

`public __construct (string $name, [array $attributes])` inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

`public Phalcon\Forms\ElementInterface setForm (Phalcon\Forms\Form $form)` inherited from Phalcon\Forms\Element

Sets the parent form to the element

`public Phalcon\Forms\ElementInterface getForm ()` inherited from Phalcon\Forms\Element

Returns the parent form to the element

`public Phalcon\Forms\ElementInterface setName (string $name)` inherited from Phalcon\Forms\Element

Sets the element's name

`public string getName ()` inherited from Phalcon\Forms\Element

Returns the element's name

`public Phalcon\Forms\ElementInterface setFilters (array/string $filters)` inherited from Phalcon\Forms\Element

Sets the element's filters

`public Phalcon\Forms\ElementInterface addFilter (string $filter)` inherited from Phalcon\Forms\Element

Adds a filter to current list of filters

`public mixed getFilters ()` inherited from Phalcon\Forms\Element

Returns the element's filters

`public Phalcon\Forms\ElementInterface addValidators (unknown $validators, [unknown $merge])` inherited from Phalcon\Forms\Element

Adds a group of validators

`public Phalcon\Forms\ElementInterface addValidator (unknown $validator)` inherited from Phalcon\Forms\Element

Adds a validator to the element

`public Phalcon\Validation\ValidatorInterface [] getValidators ()` inherited from Phalcon\Forms\Element

Returns the validators registered for the element

`public array prepareAttributes ([array $attributes], [boolean $useChecked])` inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

`public Phalcon\Forms\ElementInterface setAttribute (string $attribute, mixed $value)` inherited from Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in \$\_POST

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group) inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

```
public Phalcon\Forms\ElementInterface appendMessage (Phalcon\Validation\Message $message)
inherited from Phalcon\Forms\Element
```

Appends a message to the internal message list

```
public Phalcon\Forms\Element clear () inherited from Phalcon\Forms\Element
```

Clears every element in the form to its default value

```
public string __toString () inherited from Phalcon\Forms\Element
```

Magic method \_\_toString renders the widget without attributes

## 2.54.101 Class Phalcon\Forms\Element\File

*extends abstract class Phalcon\Forms\Element*

*implements Phalcon\Forms\ElementInterface*

Component INPUT[type=file] for forms

### Methods

```
public string render ([array $attributes])
```

Renders the element widget returning html

```
public __construct (string $name, [array $attributes]) inherited from Phalcon\Forms\Element
```

Phalcon\Forms\Element constructor

```
public Phalcon\Forms\ElementInterface setForm (Phalcon\Forms\Form $form) inherited from
Phalcon\Forms\Element
```

Sets the parent form to the element

```
public Phalcon\Forms\ElementInterface getForm () inherited from Phalcon\Forms\Element
```

Returns the parent form to the element

```
public Phalcon\Forms\ElementInterface setName (string $name) inherited from Phalcon\Forms\Element
```

Sets the element's name

```
public string getName () inherited from Phalcon\Forms\Element
```

Returns the element's name

```
public Phalcon\Forms\ElementInterface setFilters (array/string $filters) inherited from
Phalcon\Forms\Element
```

Sets the element's filters

```
public Phalcon\Forms\ElementInterface addFilter (string $filter) inherited from Phalcon\Forms\Element
```

Adds a filter to current list of filters

```
public mixed getFilters () inherited from Phalcon\Forms\Element
```

Returns the element's filters

```
public Phalcon\Forms\ElementInterface addValidators (unknown $validators, [unknown $merge])
inherited from Phalcon\Forms\Element
```

Adds a group of validators

```
public Phalcon\Forms\ElementInterface addValidator (unknown $validator) inherited from Phalcon\Forms\Element
```

Adds a validator to the element

```
public Phalcon\Validation\ValidatorInterface [] getValidators () inherited from Phalcon\Forms\Element
```

Returns the validators registered for the element

```
public array prepareAttributes ([array $attributes], [boolean $useChecked]) inherited from Phalcon\Forms\Element
```

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

```
public Phalcon\Forms\ElementInterface setAttribute (string $attribute, mixed $value) inherited from Phalcon\Forms\Element
```

Sets a default attribute for the element

```
public mixed getAttribute (string $attribute, [mixed $defaultValue]) inherited from Phalcon\Forms\Element
```

Returns the value of an attribute if present

```
public Phalcon\Forms\ElementInterface setAttributes (array $attributes) inherited from Phalcon\Forms\Element
```

Sets default attributes for the element

```
public array getAttributes () inherited from Phalcon\Forms\Element
```

Returns the default attributes for the element

```
public Phalcon\Forms\ElementInterface setUserOption (string $option, mixed $value) inherited from Phalcon\Forms\Element
```

Sets an option for the element

```
public mixed getUserOption (string $option, [mixed $defaultValue]) inherited from Phalcon\Forms\Element
```

Returns the value of an option if present

```
public Phalcon\Forms\ElementInterface setUserOptions (array $options) inherited from Phalcon\Forms\Element
```

Sets options for the element

```
public array getUserOptions () inherited from Phalcon\Forms\Element
```

Returns the options for the element

```
public Phalcon\Forms\ElementInterface setLabel (string $label) inherited from Phalcon\Forms\Element
```

Sets the element label

```
public string getLabel () inherited from Phalcon\Forms\Element
```

Returns the element's label

```
public string label (unknown $attributes) inherited from Phalcon\Forms\Element
```

Generate the HTML to label the element

```
public Phalcon\Forms\ElementInterface setDefault (mixed $value) inherited from Phalcon\Forms\Element
```

Sets a default value in case the form does not use an entity or there is no value available for the element in `$_POST`

`public mixed getDefault ()` inherited from `Phalcon\Forms\Element`

Returns the default value assigned to the element

`public mixed getValue ()` inherited from `Phalcon\Forms\Element`

Returns the element's value

`public Phalcon\Validation\Message\Group getMessages ()` inherited from `Phalcon\Forms\Element`

Returns the messages that belongs to the element The element needs to be attached to a form

`public boolean hasMessages ()` inherited from `Phalcon\Forms\Element`

Checks whether there are messages attached to the element

`public Phalcon\Forms\ElementInterface setMessages (Phalcon\Validation\Message\Group $group)`  
inherited from `Phalcon\Forms\Element`

Sets the validation messages related to the element

`public Phalcon\Forms\ElementInterface appendMessage (Phalcon\Validation\Message $message)`  
inherited from `Phalcon\Forms\Element`

Appends a message to the internal message list

`public Phalcon\Forms\Element clear ()` inherited from `Phalcon\Forms\Element`

Clears every element in the form to its default value

`public string __toString ()` inherited from `Phalcon\Forms\Element`

Magic method `__toString` renders the widget without attributes

## 2.54.102 Class `Phalcon\Forms\Element\Hidden`

*extends abstract class `Phalcon\Forms\Element`*

*implements `Phalcon\Forms\ElementInterface`*

Component INPUT[type=hidden] for forms

### Methods

`public string render ([array $attributes])`

Renders the element widget returning html

`public __construct (string $name, [array $attributes])` inherited from `Phalcon\Forms\Element`

`Phalcon\Forms\Element` constructor

`public Phalcon\Forms\ElementInterface setForm (Phalcon\Forms\Form $form)` inherited from `Phalcon\Forms\Element`

Sets the parent form to the element

`public Phalcon\Forms\ElementInterface getForm ()` inherited from `Phalcon\Forms\Element`

Returns the parent form to the element

`public Phalcon\Forms\ElementInterface setName (string $name)` inherited from `Phalcon\Forms\Element`

Sets the element's name

public *string* **getName** () inherited from Phalcon\Forms\Element

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters) inherited from Phalcon\Forms\Element

Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter) inherited from Phalcon\Forms\Element

Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from Phalcon\Forms\Element

Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge]) inherited from Phalcon\Forms\Element

Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator) inherited from Phalcon\Forms\Element

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** () inherited from Phalcon\Forms\Element

Returns the validators registered for the element

public *array* **prepareAttributes** ([*array* \$attributes], [*boolean* \$useChecked]) inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group) inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message) inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element

Magic method **\_\_toString** renders the widget without attributes

## 2.54.103 Class Phalcon\Forms\Element\Numeric

*extends* abstract class *Phalcon\Forms\Element*

*implements* *Phalcon\Forms\ElementInterface*

Component INPUT[type=number] for forms

## Methods

public *string* **render** ([*array* \$attributes])

Renders the element widget returning html

public **\_\_construct** (*string* \$name, [*array* \$attributes]) inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form) inherited from Phalcon\Forms\Element

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface* **getForm** () inherited from Phalcon\Forms\Element

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name) inherited from Phalcon\Forms\Element

Sets the element's name

public *string* **getName** () inherited from Phalcon\Forms\Element

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters) inherited from Phalcon\Forms\Element

Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter) inherited from Phalcon\Forms\Element

Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from Phalcon\Forms\Element

Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge]) inherited from Phalcon\Forms\Element

Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator) inherited from Phalcon\Forms\Element

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** () inherited from Phalcon\Forms\Element

Returns the validators registered for the element

public *array* **prepareAttributes** ([*array* \$attributes], [*boolean* \$useChecked]) inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in \$\_POST

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group) inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message) inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from *Phalcon\Forms\Element*

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from *Phalcon\Forms\Element*

Magic method **\_\_toString** renders the widget without attributes

## 2.54.104 Class *Phalcon\Forms\Element\Password*

*extends abstract class Phalcon\Forms\Element*

*implements Phalcon\Forms\ElementInterface*

Component INPUT[type=password] for forms

### Methods

public *string* **render** ([*array* \$attributes])

Renders the element widget returning html

public **\_\_construct** (*string* \$name, [*array* \$attributes]) inherited from *Phalcon\Forms\Element*

*Phalcon\Forms\Element* constructor

public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form) inherited from *Phalcon\Forms\Element*

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface* **getForm** () inherited from *Phalcon\Forms\Element*

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name) inherited from *Phalcon\Forms\Element*

Sets the element's name

public *string* **getName** () inherited from *Phalcon\Forms\Element*

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters) inherited from *Phalcon\Forms\Element*

Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter) inherited from *Phalcon\Forms\Element*

Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from *Phalcon\Forms\Element*

Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge]) inherited from *Phalcon\Forms\Element*

Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator) inherited from *Phalcon\Forms\Element*

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** () inherited from Phalcon\Forms\Element

Returns the validators registered for the element

public *array* **prepareAttributes** ([*array* \$attributes], [*boolean* \$useChecked]) inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group) inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message) inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element

Magic method \_\_toString renders the widget without attributes

## 2.54.105 Class Phalcon\Forms\Element\Radio

*extends* abstract class *Phalcon\Forms\Element*

*implements* *Phalcon\Forms\ElementInterface*

input[type="radio"] for forms

### Methods

public *string* **render** ([*array* \$attributes])

Renders the element widget returning html

public **\_\_construct** (*string* \$name, [*array* \$attributes]) inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form) inherited from Phalcon\Forms\Element

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface* **getForm** () inherited from Phalcon\Forms\Element

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name) inherited from Phalcon\Forms\Element

Sets the element's name

public *string* **getName** () inherited from Phalcon\Forms\Element

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* *\$filters*) inherited from *Phalcon\Forms\Element*  
Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* *\$filter*) inherited from *Phalcon\Forms\Element*  
Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from *Phalcon\Forms\Element*  
Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* *\$validators*, [*unknown* *\$merge*]) inherited from *Phalcon\Forms\Element*  
Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* *\$validator*) inherited from *Phalcon\Forms\Element*  
Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** () inherited from *Phalcon\Forms\Element*  
Returns the validators registered for the element

public *array* **prepareAttributes** ([*array* *\$attributes*, [*boolean* *\$useChecked*]]) inherited from *Phalcon\Forms\Element*  
Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* *\$attribute*, *mixed* *\$value*) inherited from *Phalcon\Forms\Element*  
Sets a default attribute for the element

public *mixed* **getAttribute** (*string* *\$attribute*, [*mixed* *\$defaultValue*]) inherited from *Phalcon\Forms\Element*  
Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* *\$attributes*) inherited from *Phalcon\Forms\Element*  
Sets default attributes for the element

public *array* **getAttributes** () inherited from *Phalcon\Forms\Element*  
Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* *\$option*, *mixed* *\$value*) inherited from *Phalcon\Forms\Element*  
Sets an option for the element

public *mixed* **getUserOption** (*string* *\$option*, [*mixed* *\$defaultValue*]) inherited from *Phalcon\Forms\Element*  
Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* *\$options*) inherited from *Phalcon\Forms\Element*  
Sets options for the element

public *array* **getUserOptions** () inherited from *Phalcon\Forms\Element*

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group)  
inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message)  
inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element

Magic method **\_\_toString** renders the widget without attributes

## 2.54.106 Class Phalcon\Forms\Element\Select

*extends* abstract class *Phalcon\Forms\Element*

*implements* *Phalcon\Forms\ElementInterface*

Component SELECT (choice) for forms

## Methods

public **\_\_construct** (*string \$name*, [*object/array \$options*], [*array \$attributes*])  
 Phalcon\Forms\Element constructor

public *Phalcon\Forms\Element setOptions* (*array/object \$options*)  
 Set the choice's options

public *array/object getOptions* ()  
 Returns the choices' options

public *\$this addOption* (*array \$option*)  
 Adds an option to the current options

public *string render* ([*array \$attributes*])  
 Renders the element widget returning html

public *Phalcon\Forms\ElementInterface setForm* (*Phalcon\Forms\Form \$form*) inherited from Phalcon\Forms\Element  
 Sets the parent form to the element

public *Phalcon\Forms\ElementInterface getForm* () inherited from Phalcon\Forms\Element  
 Returns the parent form to the element

public *Phalcon\Forms\ElementInterface setName* (*string \$name*) inherited from Phalcon\Forms\Element  
 Sets the element's name

public *string getName* () inherited from Phalcon\Forms\Element  
 Returns the element's name

public *Phalcon\Forms\ElementInterface setFilters* (*array/string \$filters*) inherited from Phalcon\Forms\Element  
 Sets the element's filters

public *Phalcon\Forms\ElementInterface addFilter* (*string \$filter*) inherited from Phalcon\Forms\Element  
 Adds a filter to current list of filters

public *mixed getFilters* () inherited from Phalcon\Forms\Element  
 Returns the element's filters

public *Phalcon\Forms\ElementInterface addValidators* (*unknown \$validators*, [*unknown \$merge*]) inherited from Phalcon\Forms\Element  
 Adds a group of validators

public *Phalcon\Forms\ElementInterface addValidator* (*unknown \$validator*) inherited from Phalcon\Forms\Element  
 Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface [] getValidators* () inherited from Phalcon\Forms\Element  
 Returns the validators registered for the element

public *array prepareAttributes* (*[array \$attributes*, [*boolean \$useChecked*]]) inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element. The element needs to be attached to a form

public *boolean hasMessages ()* inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface setMessages (Phalcon\Validation\Message\Group \$group)*  
inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface appendMessage (Phalcon\Validation\Message \$message)*  
inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element clear ()* inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string \_\_toString ()* inherited from Phalcon\Forms\Element

Magic method \_\_toString renders the widget without attributes

## 2.54.107 Class Phalcon\Forms\Element\Submit

*extends abstract class Phalcon\Forms\Element*

*implements Phalcon\Forms\ElementInterface*

Component INPUT[type=submit] for forms

### Methods

public *string render ([array \$attributes])*

Renders the element widget

public *\_\_construct (string \$name, [array \$attributes])* inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface setForm (Phalcon\Forms\Form \$form)* inherited from Phalcon\Forms\Element

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface getForm ()* inherited from Phalcon\Forms\Element

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface setName (string \$name)* inherited from Phalcon\Forms\Element

Sets the element's name

public *string getName ()* inherited from Phalcon\Forms\Element

Returns the element's name

public *Phalcon\Forms\ElementInterface setFilters (array/string \$filters)* inherited from Phalcon\Forms\Element

Sets the element's filters

public *Phalcon\Forms\ElementInterface addFilter (string \$filter)* inherited from Phalcon\Forms\Element

Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from Phalcon\Forms\Element

Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge])  
inherited from Phalcon\Forms\Element

Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator) inherited from  
Phalcon\Forms\Element

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** () inherited from Phalcon\Forms\Element

Returns the validators registered for the element

public *array* **prepareAttributes** ([*array* \$attributes], [*boolean* \$useChecked]) inherited from  
Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from  
Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from  
Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from  
Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from  
Phalcon\Forms\Element

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from  
Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from  
Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element  
 Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element  
 Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element  
 Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element  
 Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element  
 Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element  
 Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group)  
 inherited from Phalcon\Forms\Element  
 Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message)  
 inherited from Phalcon\Forms\Element  
 Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element  
 Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element  
 Magic method **\_\_toString** renders the widget without attributes

## 2.54.108 Class Phalcon\Forms\Element\Text

*extends* abstract class *Phalcon\Forms\Element*

*implements* *Phalcon\Forms\ElementInterface*

Component INPUT[type=text] for forms

### Methods

public *string* **render** ([*array* \$attributes])

Renders the element widget

public **\_\_construct** (*string* \$name, [*array* \$attributes]) inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form) inherited from Phalcon\Forms\Element  
 Sets the parent form to the element

public *Phalcon\Forms\ElementInterface* **getForm** () inherited from Phalcon\Forms\Element

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name) inherited from Phalcon\Forms\Element

Sets the element's name

public *string* **getName** () inherited from Phalcon\Forms\Element

Returns the element's name

public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters) inherited from Phalcon\Forms\Element

Sets the element's filters

public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter) inherited from Phalcon\Forms\Element

Adds a filter to current list of filters

public *mixed* **getFilters** () inherited from Phalcon\Forms\Element

Returns the element's filters

public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge]) inherited from Phalcon\Forms\Element

Adds a group of validators

public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator) inherited from Phalcon\Forms\Element

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface* [] **getValidators** () inherited from Phalcon\Forms\Element

Returns the validators registered for the element

public *array* **prepareAttributes** ([*array* \$attributes], [*boolean* \$useChecked]) inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from Phalcon\Forms\Element

Sets default attributes for the element

public *array* **getAttributes** () inherited from Phalcon\Forms\Element

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from Phalcon\Forms\Element

Sets an option for the element

---

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from Phalcon\Forms\Element

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from Phalcon\Forms\Element

Sets options for the element

public *array* **getUserOptions** () inherited from Phalcon\Forms\Element

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from Phalcon\Forms\Element

Sets the element label

public *string* **getLabel** () inherited from Phalcon\Forms\Element

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from Phalcon\Forms\Element

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from Phalcon\Forms\Element

Sets a default value in case the form does not use an entity or there is no value available for the element in \$\_POST

public *mixed* **getDefault** () inherited from Phalcon\Forms\Element

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from Phalcon\Forms\Element

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from Phalcon\Forms\Element

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from Phalcon\Forms\Element

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group) inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message) inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element

Magic method \_\_toString renders the widget without attributes

## 2.54.109 Class Phalcon\Forms\Element\TextArea

*extends abstract class Phalcon\Forms\Element*

*implements Phalcon\Forms\ElementInterface*

Component TEXTAREA for forms

### Methods

public *string render ([array \$attributes])*

Renders the element widget

public *\_\_construct (string \$name, [array \$attributes])* inherited from Phalcon\Forms\Element

Phalcon\Forms\Element constructor

public *Phalcon\Forms\ElementInterface setForm (Phalcon\Forms\Form \$form)* inherited from Phalcon\Forms\Element

Sets the parent form to the element

public *Phalcon\Forms\ElementInterface getForm ()* inherited from Phalcon\Forms\Element

Returns the parent form to the element

public *Phalcon\Forms\ElementInterface setName (string \$name)* inherited from Phalcon\Forms\Element

Sets the element's name

public *string getName ()* inherited from Phalcon\Forms\Element

Returns the element's name

public *Phalcon\Forms\ElementInterface setFilters (array/string \$filters)* inherited from Phalcon\Forms\Element

Sets the element's filters

public *Phalcon\Forms\ElementInterface addFilter (string \$filter)* inherited from Phalcon\Forms\Element

Adds a filter to current list of filters

public *mixed getFilters ()* inherited from Phalcon\Forms\Element

Returns the element's filters

public *Phalcon\Forms\ElementInterface addValidators (unknown \$validators, [unknown \$merge])* inherited from Phalcon\Forms\Element

Adds a group of validators

public *Phalcon\Forms\ElementInterface addValidator (unknown \$validator)* inherited from Phalcon\Forms\Element

Adds a validator to the element

public *Phalcon\Validation\ValidatorInterface [] getValidators ()* inherited from Phalcon\Forms\Element

Returns the validators registered for the element

public *array prepareAttributes ([array \$attributes], [boolean \$useChecked])* inherited from Phalcon\Forms\Element

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value) inherited from *Phalcon\Forms\Element*

Sets a default attribute for the element

public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue]) inherited from *Phalcon\Forms\Element*

Returns the value of an attribute if present

public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes) inherited from *Phalcon\Forms\Element*

Sets default attributes for the element

public *array* **getAttributes** () inherited from *Phalcon\Forms\Element*

Returns the default attributes for the element

public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value) inherited from *Phalcon\Forms\Element*

Sets an option for the element

public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue]) inherited from *Phalcon\Forms\Element*

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options) inherited from *Phalcon\Forms\Element*

Sets options for the element

public *array* **getUserOptions** () inherited from *Phalcon\Forms\Element*

Returns the options for the element

public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label) inherited from *Phalcon\Forms\Element*

Sets the element label

public *string* **getLabel** () inherited from *Phalcon\Forms\Element*

Returns the element's label

public *string* **label** (*unknown* \$attributes) inherited from *Phalcon\Forms\Element*

Generate the HTML to label the element

public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value) inherited from *Phalcon\Forms\Element*

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

public *mixed* **getDefault** () inherited from *Phalcon\Forms\Element*

Returns the default value assigned to the element

public *mixed* **getValue** () inherited from *Phalcon\Forms\Element*

Returns the element's value

public *Phalcon\Validation\Message\Group* **getMessages** () inherited from *Phalcon\Forms\Element*

Returns the messages that belongs to the element The element needs to be attached to a form

public *boolean* **hasMessages** () inherited from *Phalcon\Forms\Element*

Checks whether there are messages attached to the element

public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message|Group* \$group)  
inherited from Phalcon\Forms\Element

Sets the validation messages related to the element

public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message)  
inherited from Phalcon\Forms\Element

Appends a message to the internal message list

public *Phalcon\Forms\Element* **clear** () inherited from Phalcon\Forms\Element

Clears every element in the form to its default value

public *string* **\_\_toString** () inherited from Phalcon\Forms\Element

Magic method **\_\_toString** renders the widget without attributes

## 2.54.110 Class Phalcon\Forms\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Forms will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

### 2.54.111 Class Phalcon\Forms\Form

*extends abstract class Phalcon\DI\Injectable*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\DI\InjectionAwareInterface, Countable, Iterator, Traversable*

This component allows to build forms using an object-oriented interface

#### Methods

public **\_\_construct** ([*object \$entity*], [*array \$userOptions*])

Phalcon\Forms\Form constructor

public *Phalcon\Forms\Form* **setAction** (*string \$action*)

Sets the form's action

public *string* **getAction** ()

Returns the form's action

public *Phalcon\Forms\Form* **setUserOption** (*string \$option, mixed \$value*)

Sets an option for the form

public *mixed* **getUserOption** (*string \$option, [mixed \$defaultValue]*)

Returns the value of an option if present

public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array \$options*)

Sets options for the element

public *array* **getUserOptions** ()

Returns the options for the element

public *Phalcon\Forms\Form* **setEntity** (*object \$entity*)

Sets the entity related to the model

public *object* **getEntity** ()

Returns the entity related to the model

public *Phalcon\Forms\ElementInterface []* **getElements** ()

Returns the form elements added to the form

public *Phalcon\Forms\Form* **bind** (*array \$data, object \$entity, [array \$whitelist]*)

Binds data to the entity

public *boolean* **isValid** (*[array \$data], [object \$entity]*)

Validates the form

public *Phalcon\Validation\Message\Group* **getMessages** (*[boolean \$byItemName]*)

Returns the messages generated in the validation

public *Phalcon\Validation\Message\Group []* **getMessagesFor** (*unknown \$name*)

Returns the messages generated for a specific element

`public boolean hasMessagesFor (unknown $name)`

Check if messages were generated for a specific element

`public Phalcon\Forms\Form add (Phalcon\Forms\ElementInterface $element, [string $postion], [unknown $type])`

Adds an element to the form

`public string render (string $name, [array $attributes])`

Renders a specific item in the form

`public Phalcon\Forms\ElementInterface get (string $name)`

Returns an element added to the form by its name

`public string label (string $name, [unknown $attributes])`

Generate the label of a element added to the form including HTML

`public string getLabel (string $name)`

Returns a label for an element

`public mixed getValue (string $name)`

Gets a value from the internal related entity or from the default value

`public boolean has (string $name)`

Check if the form contains an element

`public boolean remove (string $name)`

Removes an element from the form

`public Phalcon\Forms\Form clear ([array $fields])`

Clears every element in the form to its default value

`public int count ()`

Returns the number of elements in the form

`public rewind ()`

Rewinds the internal iterator

`public Phalcon\Validation\Message current ()`

Returns the current element in the iterator

`public int key ()`

Returns the current position/key in the iterator

`public next ()`

Moves the internal iteration pointer to the next position

`public boolean valid ()`

Check if the current element in the iterator is valid

`public setDI (Phalcon\DiInterface $dependencyInjector) inherited from Phalcon\DI\Injectable`

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\DI\Injectable

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\DI\Injectable

Returns the internal event manager

public **\_\_get** (*unknown* \$property) inherited from Phalcon\DI\Injectable

Magic method \_\_get

## 2.54.112 Class Phalcon\Forms\Manager

Manages forms within the application. Allowing the developer to access them from any part of the application

### Methods

public **\_\_construct** ()

...

public *Phalcon\Forms\Form* **create** ([*string* \$name], [*object* \$entity])

Creates a form registering it in the forms manager

public *Phalcon\Forms\Form* **get** (*string* \$name)

Returns a form by its name

public *boolean* **has** (*string* \$name)

Checks if a form is registered in the forms manager

public *Phalcon\Forms\Manager* **set** (*string* \$name, *Phalcon\Forms\Form* \$form)

Registers a form in the Forms Manager

## 2.54.113 Class Phalcon\Http\Cookie

*implements Phalcon\DI\InjectionAwareInterface*

Provide OO wrappers to manage a HTTP cookie

### Methods

public **\_\_construct** (*string* \$name, [*mixed* \$value], [*int* \$expire], [*string* \$path], [*boolean* \$secure], [*string* \$domain], [*boolean* \$httpOnly])

Phalcon\Http\Cookie constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

`public Phalcon\Http\CookieInterface setValue (string $value)`

Sets the cookie's value

`public mixed getValue ([string/array $filters], [string $defaultValue])`

Returns the cookie's value

`public Phalcon\Http\Cookie send ()`

Sends the cookie to the HTTP client Stores the cookie definition in session

`public Phalcon\Http\Cookie restore ()`

Reads the cookie-related info from the SESSION to restore the cookie as it was set This method is automatically called internally so normally you don't need to call it

`public delete ()`

Deletes the cookie by setting an expire time in the past

`public Phalcon\Http\Cookie useEncryption (boolean $useEncryption)`

Sets if the cookie must be encrypted/decrypted automatically

`public boolean isUsingEncryption ()`

Check if the cookie is using implicit encryption

`public Phalcon\Http\Cookie setExpiration (int $expire)`

Sets the cookie's expiration time

`public string getExpiration ()`

Returns the current expiration time

`public Phalcon\Http\Cookie setPath (string $path)`

Sets the cookie's expiration time

`public string getPath ()`

Returns the current cookie's path

`public Phalcon\Http\Cookie setDomain (string $domain)`

Sets the domain that the cookie is available to

`public string getDomain ()`

Returns the domain that the cookie is available to

`public Phalcon\Http\Cookie setSecure (boolean $secure)`

Sets if the cookie must only be sent when the connection is secure (HTTPS)

`public boolean getSecure ()`

Returns whether the cookie must only be sent when the connection is secure (HTTPS)

`public Phalcon\Http\Cookie setHttpOnly (boolean $httpOnly)`

Sets if the cookie is accessible only through the HTTP protocol

`public boolean getHttpOnly ()`

Returns if the cookie is accessible only through the HTTP protocol

public *mixed* **\_\_toString** ()

Magic \_\_toString method converts the cookie's value to string

### 2.54.114 Class Phalcon\Http\Cookie\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Http\Cookie will use this class

#### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

### 2.54.115 Class Phalcon\Http\Request

*implements Phalcon\Http\RequestInterface, Phalcon\DI\InjectionAwareInterface*

Encapsulates request information for easy and secure access from application controllers. The request object is a simple value object that is passed between the dispatcher and controller classes. It packages the HTTP request environment.

```
<?php

$request = new Phalcon\Http\Request();
if ($request->isPost() == true) {
    if ($request->isAjax() == true) {
        echo 'Request was made using POST and AJAX';
    }
}
```

## Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

public *mixed* **get** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets a variable from the `$_REQUEST` superglobal applying filters if needed. If no parameters are given the `$_REQUEST` superglobal is returned

```
<?php
```

```
//Returns value from $_REQUEST["user_email"] without sanitizing
$userEmail = $request->get("user_email");
```

```
//Returns value from $_REQUEST["user_email"] with sanitizing
$userEmail = $request->get("user_email", "email");
```

public *mixed* **getPost** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets a variable from the `$_POST` superglobal applying filters if needed. If no parameters are given the `$_POST` superglobal is returned

```
<?php
```

```
//Returns value from $_POST["user_email"] without sanitizing
$userEmail = $request->getPost("user_email");
```

```
//Returns value from $_POST["user_email"] with sanitizing
$userEmail = $request->getPost("user_email", "email");
```

public *mixed* **getPut** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets a variable from put request

```
<?php
```

```
$userEmail = $request->getPut("user_email");
```

```
$userEmail = $request->getPut("user_email", "email");
```

public *mixed* **getQuery** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets variable from `$_GET` superglobal applying filters if needed. If no parameters are given the `$_GET` superglobal is returned

```
<?php  
  
//Returns value from $_GET["id"] without sanitizing  
$id = $request->getQuery("id");
```

```
//Returns value from $_GET["id"] with sanitizing  
$id = $request->getQuery("id", "int");  
  
//Returns value from $_GET["id"] with a default value  
$id = $request->getQuery("id", null, 150);
```

**public mixed getServer (string \$name)**

Gets variable from \$\_SERVER superglobal

**public boolean has (string \$name)**

Checks whether \$\_REQUEST superglobal has certain index

**public boolean hasPost (string \$name)**

Checks whether \$\_POST superglobal has certain index

**public boolean hasPut (string \$name)**

Checks whether put has certain index

**public boolean hasQuery (string \$name)**

Checks whether \$\_GET superglobal has certain index

**public mixed hasServer (string \$name)**

Checks whether \$\_SERVER superglobal has certain index

**public string getHeader (string \$header)**

Gets HTTP header from request data

**public string getScheme ()**

Gets HTTP schema (http/https)

**public boolean isAjax ()**

Checks whether request has been made using ajax. Checks if \$\_SERVER['HTTP\_X\_REQUESTED\_WITH']=='XMLHttpRequest'

**public boolean isSoapRequested ()**

Checks whether request has been made using SOAP

**public boolean isSecureRequest ()**

Checks whether request has been made using any secure layer

**public string getRawBody ()**

Gets HTTP raw request body

**public string getJsonRawBody ()**

Gets decoded JSON HTTP raw request body

**public string getServerAddress ()**

Gets active server address IP

**public string getServerName ()**

Gets active server name

**public string getHttpHost ()**

Gets information about schema, host and port used by the request

**public string getClientAddress ([boolean \$trustForwardedHeader])**

Gets most possible client IPv4 Address. This method search in `$_SERVER['REMOTE_ADDR']` and optionally in `$_SERVER['HTTP_X_FORWARDED_FOR']`

**public string getMethod ()**

Gets HTTP method which request has been made

**public string getURI ()**

Gets HTTP URI which request has been made

**public string getUserAgent ()**

Gets HTTP user agent used to made the request

**public boolean isMethod (string/array \$methods)**

Check if HTTP method match any of the passed methods

**public boolean isPost ()**

Checks whether HTTP method is POST. if `$_SERVER['REQUEST_METHOD']=='POST'`

**public boolean isGet ()**

Checks whether HTTP method is GET. if `$_SERVER['REQUEST_METHOD']=='GET'`

**public boolean isPut ()**

Checks whether HTTP method is PUT. if `$_SERVER['REQUEST_METHOD']=='PUT'`

**public boolean isPatch ()**

Checks whether HTTP method is PATCH. if `$_SERVER['REQUEST_METHOD']=='PATCH'`

**public boolean isHead ()**

Checks whether HTTP method is HEAD. if `$_SERVER['REQUEST_METHOD']=='HEAD'`

**public boolean isDelete ()**

Checks whether HTTP method is DELETE. if `$_SERVER['REQUEST_METHOD']=='DELETE'`

**public boolean isOptions ()**

Checks whether HTTP method is OPTIONS. if `$_SERVER['REQUEST_METHOD']=='OPTIONS'`

**public boolean hasFiles ([unknown \$notErrored])**

Checks whether request includes attached files

**public Phalcon\Http\Request\File [] getUploadedFiles ([boolean \$notErrored])**

Gets attached files as Phalcon\Http\Request\File instances

**public array getHeaders ()**

Returns the available headers in the request

**public string getHTTPReferer ()**

Gets web page that refers active request. ie: <http://www.google.com>

```

protected array _getQualityHeader ()
Process a request header and return an array of values with their qualities

protected string _getBestQuality ()
Process a request header and return the one with best quality

public array getAcceptableContent ()
Gets array with mime/types and their quality accepted by the browser/client from
$_SERVER['HTTP_ACCEPT']

public array getBestAccept ()
Gets best mime/type accepted by the browser/client from $_SERVER['HTTP_ACCEPT']

public array getClientCharsets ()
Gets charsets array and their quality accepted by the browser/client from
$_SERVER['HTTP_ACCEPT_CHARSET']

public string getBestCharset ()
Gets best charset accepted by the browser/client from $_SERVER['HTTP_ACCEPT_CHARSET']

public array getLanguages ()
Gets languages array and their quality accepted by the browser/client from
$_SERVER['HTTP_ACCEPT_LANGUAGE']

public string getBestLanguage ()
Gets best language accepted by the browser/client from $_SERVER['HTTP_ACCEPT_LANGUAGE']

public array getBasicAuth ()
Gets auth info accepted by the browser/client from $_SERVER['PHP_AUTH_USER']

public array getDigestAuth ()
Gets auth info accepted by the browser/client from $_SERVER['PHP_AUTH_DIGEST']

```

## 2.54.116 Class Phalcon\Http\Request\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Http\Request will use this class

### Methods

```

final private Exception __clone () inherited from Exception
Clone the exception

public __construct ([string $message], [int $code], [Exception $previous]) inherited from Exception
Exception constructor

final public string getMessage () inherited from Exception
Gets the Exception message

final public int getCode () inherited from Exception
Gets the Exception code

```

final public *string* **getFile** () inherited from Exception  
Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception  
Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception  
Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception  
Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception  
Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception  
String representation of the exception

## 2.54.117 Class Phalcon\Http\Request\File

*extends* SplFileInfo

*implements* Phalcon\Http\Request\FileInterface

Provides OO wrappers to the \$\_FILES superglobal

<?php

```
class PostsController extends \Phalcon\Mvc\Controller
{

    public function uploadAction()
    {
        //Check if the user has uploaded files
        if ($this->request->hasFiles() == true) {
            //Print the real file names and their sizes
            foreach ($this->request->getUploadedFiles() as $file){
                echo $file->getName(), " ", $file->getSize(), "\n";
            }
        }
    }
}
```

### Methods

public **\_\_construct** (*array* \$file)

Phalcon\Http\Request\File constructor

public *int* **getSize** ()

Returns the file size of the uploaded file

public *string* **getName** ()

Returns the real name of the uploaded file

public *string* **getTempName** ()

Returns the temporary name of the uploaded file

public *string* **getType** ()

Returns the mime type reported by the browser This mime type is not completely secure, use `getRealType()` instead

public *string* **getRealType** ()

Gets the real mime type of the upload file using finfo

public *string* **getError** ()

Returns the error code

public *string* **getKey** ()

Returns the file key

public *boolean* **isUploadedFile** ()

Checks whether the file has been uploaded via Post.

public *boolean* **moveTo** (*string* \$destination)

Moves the temporary file to a destination within the application

public static *\_\_set\_state* (*unknown* \$params)

...

public *string* **getExtension** ()

Returns the file extension

public **getPath** () inherited from `SplFileInfo`

...

public **getFilename** () inherited from `SplFileInfo`

...

public **getBasename** ([*unknown* \$suffix]) inherited from `SplFileInfo`

...

public **getPathname** () inherited from `SplFileInfo`

...

public **getPerms** () inherited from `SplFileInfo`

...

public **getInode** () inherited from `SplFileInfo`

...

public **getOwner** () inherited from `SplFileInfo`

...

public **getGroup** () inherited from `SplFileInfo`

...

public **getATime** () inherited from `SplFileInfo`

...

public **getMTime** () inherited from SplFileInfo

...

public **getCTime** () inherited from SplFileInfo

...

public **isWritable** () inherited from SplFileInfo

...

public **isReadable** () inherited from SplFileInfo

...

public **isExecutable** () inherited from SplFileInfo

...

public **isFile** () inherited from SplFileInfo

...

public **isDir** () inherited from SplFileInfo

...

public **isLink** () inherited from SplFileInfo

...

public **getLinkTarget** () inherited from SplFileInfo

...

public **getRealPath** () inherited from SplFileInfo

...

public **getFileInfo** ([*unknown \$class\_name*]) inherited from SplFileInfo

...

public **getPathInfo** ([*unknown \$class\_name*]) inherited from SplFileInfo

...

public **openFile** ([*unknown \$open\_mode*], [*unknown \$use\_include\_path*], [*unknown \$context*]) inherited from SplFileInfo

...

public **setFileClass** ([*unknown \$class\_name*]) inherited from SplFileInfo

...

public **setInfoClass** ([*unknown \$class\_name*]) inherited from SplFileInfo

...

final public **\_bad\_state\_ex** () inherited from SplFileInfo

...

public **\_\_toString** () inherited from SplFileInfo

...

## 2.54.118 Class Phalcon\Http\Response

*implements Phalcon\Http\ResponseInterface, Phalcon\DI\InjectionAwareInterface*

Part of the HTTP cycle is return responses to the clients. Phalcon\HTTP\Response is the Phalcon component responsible to achieve this task. HTTP responses are usually composed by headers and body.

<?php

```
$response = new Phalcon\Http\Response();
$response->setStatusCode(200, "OK");
$response->setContent("<html><body>Hello</body></html>");
$response->send();
```

### Methods

public **\_\_construct** ([*string* \$content], [*int* \$code], [*string* \$status])

Phalcon\Http\Response constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

public *Phalcon\Http\ResponseInterface* **setStatusCode** (*int* \$code, *string* \$message)

Sets the HTTP response code

<?php

```
$response->setStatusCode(404, "Not Found");
```

public *Phalcon\Http\ResponseInterface* **setHeaders** (*Phalcon\Http\Response\HeadersInterface* \$headers)

Sets a headers bag for the response externally

public *Phalcon\Http\Response\HeadersInterface* **getHeaders** ()

Returns headers set by the user

public *Phalcon\Http\ResponseInterface* **setCookies** (*Phalcon\Http\Response\CookiesInterface* \$cookies)

Sets a cookies bag for the response externally

public *Phalcon\Http\Response\CookiesInterface* **getCookies** ()

Returns cookies set by the user

public *Phalcon\Http\ResponseInterface* **setHeader** (*string* \$name, *string* \$value)

Overwrites a header in the response

<?php

```
$response->setHeader("Content-Type", "text/plain");
```

public *Phalcon\Http\ResponseInterface* **setRawHeader** (*string* \$header)

Send a raw header to the response

```
<?php
```

```
$response->setRawHeader("HTTP/1.1 404 Not Found");
```

```
public Phalcon\Http\ResponseInterface resetHeaders ()
```

Resets all the stablished headers

```
public Phalcon\Http\ResponseInterface setExpires (DateTime $datetime)
```

Sets a Expires header to use HTTP cache

```
<?php
```

```
$this->response->setExpires(new DateTime());
```

```
public Phalcon\Http\ResponseInterface setNotModified ()
```

Sends a Not-Modified response

```
public Phalcon\Http\ResponseInterface setContentType (string $contentType, [string $charset])
```

Sets the response content-type mime, optionally the charset

```
<?php
```

```
$response->setContentType('application/pdf');
```

```
$response->setContentType('text/plain', 'UTF-8');
```

```
public setEtag (string $etag)
```

Set a custom ETag

```
<?php
```

```
$response->setEtag(md5(time()));
```

```
public Phalcon\Http\ResponseInterface redirect ([string/array $location], [boolean $externalRedirect], [int $statusCode])
```

Redirect by HTTP to another action or URL

```
<?php
```

```
//Using a string redirect (internal/external)
```

```
$response->redirect("posts/index");
$response->redirect("http://en.wikipedia.org", true);
$response->redirect("http://www.example.com/new-location", true, 301);
```

```
//Making a redirection based on a named route
```

```
$response->redirect(array(
    "for" => "index-lang",
    "lang" => "jp",
    "controller" => "index"
));
```

```
public Phalcon\Http\ResponseInterface setContent (string $content)
```

Sets HTTP response body

```
<?php
```

```
$response->setContent("<h1>Hello!</h1>");
```

public *Phalcon\Http\ResponseInterface* **setJsonContent** (*string* \$content)

Sets HTTP response body. The parameter is automatically converted to JSON

<?php

```
$response->setJsonContent(array("status" => "OK"));
$response->setJsonContent(array("status" => "OK"), JSON_NUMERIC_CHECK);
```

•

public *Phalcon\Http\ResponseInterface* **appendContent** (*string* \$content)

Appends a string to the HTTP response body

public *string* **getContent** ()

Gets the HTTP response body

public *boolean* **isSent** ()

Check if the response is already sent

public *Phalcon\Http\ResponseInterface* **sendHeaders** ()

Sends headers to the client

public *Phalcon\Http\ResponseInterface* **sendCookies** ()

Sends cookies to the client

public *Phalcon\Http\ResponseInterface* **send** ()

Prints out HTTP response to the client

public **setFileToSend** (*string* \$filePath, [*string* \$attachmentName])

Sets an attached file to be sent at the end of the request

## 2.54.119 Class Phalcon\Http\Response\Cookies

*implements Phalcon\Http\Response\ CookiesInterface, Phalcon\DI\InjectionAwareInterface*

This class is a bag to manage the cookies A cookies bag is automatically registered as part of the ‘response’ service in the DI

### Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

public *Phalcon\Http\Response\ Cookies* **useEncryption** (*boolean* \$useEncryption)

Set if cookies in the bag must be automatically encrypted/decrypted

public *boolean* **isUsingEncryption** ()

Returns if the bag is automatically encrypting/decrypting cookies

```
public Phalcon\Http\Response\Cookie set (string $name, [mixed $value], [int $expire], [string $path], [boolean $secure], [string $domain], [boolean $httpOnly])
```

Sets a cookie to be sent at the end of the request This method overrides any cookie set before with the same name

```
public Phalcon\Http\Cookie get (string $name)
```

Gets a cookie from the bag

```
public boolean has (string $name)
```

Check if a cookie is defined in the bag or exists in the \$\_COOKIE superglobal

```
public boolean delete (string $name)
```

Deletes a cookie by its name This method does not removes cookies from the \$\_COOKIE superglobal

```
public boolean send ()
```

Sends the cookies to the client Cookies aren't sent if headers are sent in the current request

```
public Phalcon\Http\Response\Cookie reset ()
```

Reset set cookies

## 2.54.120 Class Phalcon\Http\Response\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Http\Response will use this class

### Methods

```
final private Exception __clone () inherited from Exception
```

Clone the exception

```
public __construct ([string $message], [int $code], [Exception $previous]) inherited from Exception
```

Exception constructor

```
final public string getMessage () inherited from Exception
```

Gets the Exception message

```
final public int getCode () inherited from Exception
```

Gets the Exception code

```
final public string getFile () inherited from Exception
```

Gets the file in which the exception occurred

```
final public int getLine () inherited from Exception
```

Gets the line in which the exception occurred

```
final public array getTrace () inherited from Exception
```

Gets the stack trace

```
final public Exception getPrevious () inherited from Exception
```

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from *Exception*  
 Gets the stack trace as a string  
 public *string* **\_\_toString** () inherited from *Exception*  
 String representation of the exception

## 2.54.121 Class Phalcon\Http\Response\Headers

*implements Phalcon\Http\Response\HeadersInterface*

This class is a bag to manage the response headers

### Methods

public **set** (*string* \$name, *string* \$value)  
 Sets a header to be sent at the end of the request  
 public *string* **get** (*string* \$name)  
 Gets a header value from the internal bag  
 public **setRaw** (*string* \$header)  
 Sets a raw header to be sent at the end of the request  
 public **remove** (*unknown* \$header\_index)  
 Removes a header to be sent at the end of the request  
 public *boolean* **send** ()  
 Sends the headers to the client  
 public **reset** ()  
 Reset set headers  
 public *array* **toArray** ()  
 Returns the current headers as an array  
 public static *Phalcon\Http\Response\Headers* **\_\_set\_state** (*array* \$data)  
 Restore a Phalcon\Http\Response\Headers object

## 2.54.122 Abstract class Phalcon\Image

Image manipulation support. Allows images to be resized, cropped, etc.

<?php

```
$image = new Phalcon\Image\Adapter\GD("upload/test.jpg");
$image->resize(200, 200);
$image->save();
```

## Constants

*integer* **NONE**  
*integer* **WIDTH**  
*integer* **HEIGHT**  
*integer* **AUTO**  
*integer* **INVERSE**  
*integer* **PRECISE**  
*integer* **TENSILE**  
*integer* **HORIZONTAL**  
*integer* **VERTICAL**  
*integer* **GD**  
*integer* **IMAGICK**

## 2.54.123 Abstract class Phalcon\Image\Adapter

*implements* *Phalcon\Image\AdapterInterface*

Base class for Phalcon\Image adapters

## Methods

*public string* **getRealPath** ()

Returns the real path of the image file

*public int* **getWidth** ()

Returns the width of images

*public int* **getHeight** ()

Returns the height of images

*public int* **getType** ()

Returns the type of images

*public string* **getMime** ()

Returns the mime of images

*public resource* **getImage** ()

Returns the image of images

*public Phalcon\Image\Adapter* **resize** ([*unknown* \$width], [*unknown* \$height], [*unknown* \$master])

Resize the image to the given size. Either the width or the height can be omitted and the image will be resized proportionally.

*public Phalcon\Image\Adapter* **liquidRescale** (*unknown* \$width, *unknown* \$height, [*unknown* \$delta\_x], [*unknown* \$rigidity])

This method scales the images using liquid rescaling method. Only support Imagick

```
public Phalcon\Image\Adapter crop (unknown $width, unknown $height, [unknown $offset_x], [unknown $offset_y])
```

Crop an image to the given size. Either the width or the height can be omitted and the current width or height will be used.

```
public Phalcon\Image\Adapter rotate (unknown $degrees)
```

Rotate the image by a given amount.

```
public Phalcon\Image\Adapter flip (unknown $direction)
```

Flip the image along the horizontal or vertical axis.

```
public Phalcon\Image\Adapter sharpen (unknown $amount)
```

Sharpen the image by a given amount.

```
public Phalcon\Image\Adapter reflection ([unknown $height], [unknown $opacity], [unknown $fade_in])
```

Add a reflection to an image. The most opaque part of the reflection will be equal to the opacity setting and fade out to full transparent. Alpha transparency is preserved.

```
public Phalcon\Image\AdapterInterface watermark (unknown $watermark, [unknown $offset_x], [unknown $offset_y], [unknown $opacity])
```

Add a watermark to an image with a specified opacity. Alpha transparency will be preserved.

```
public Phalcon\Image\Adapter text (unknown $text, [unknown $offset_x], [unknown $offset_y], [unknown $opacity], [unknown $color], [unknown $size], [unknown $fontfile])
```

Add a text to an image with a specified opacity.

```
public Phalcon\Image\Adapter mask (unknown $mask)
```

Composite one image onto another

```
public Phalcon\Image\Adapter background (unknown $color, [unknown $quality])
```

Set the background color of an image. This is only useful for images with alpha transparency.

```
public Phalcon\Image\Adapter blur ([unknown $radius])
```

Blur image

```
public Phalcon\Image\Adapter pixelate ([unknown $amount])
```

Pixelate image

```
public boolean save ([unknown $file], [unknown $quality])
```

Save the image. If the filename is omitted, the original image will be overwritten.

```
public Phalcon\Image\Adapter render ([unknown $type], [unknown $quality])
```

Render the image and return the binary string.

```
abstract protected _resize (unknown $width, unknown $height)
```

...

```
abstract protected _liquidRescale (unknown $width, unknown $height, unknown $delta_x, unknown $regidity)
```

...

```
abstract protected _crop (unknown $width, unknown $height, unknown $offset_x, unknown $offset_y)
```

...

```
abstract protected _rotate (unknown $degrees)
...
abstract protected _flip (unknown $direction)
...
abstract protected _sharpen (unknown $amount)
...
abstract protected _reflection (unknown $height, unknown $opacity, unknown $fade_in)
...
abstract protected _watermark (unknown $watermark, unknown $offset_x, unknown $offset_y, unknown $opacity)
...
abstract protected _text (unknown $text, unknown $offset_x, unknown $offset_y, unknown $opacity, unknown $r, unknown $g, unknown $b, unknown $size, unknown $fontfile)
...
abstract protected _mask (unknown $mask)
...
abstract protected _background (unknown $r, unknown $g, unknown $b, unknown $opacity)
...
abstract protected _blur (unknown $radius)
...
abstract protected _pixelate (unknown $amount)
...
abstract protected _save (unknown $file, unknown $quality)
...
abstract protected _render (unknown $type, unknown $quality)
...
```

## 2.54.124 Class Phalcon\Image\Adapter\GD

*extends abstract class Phalcon\Image\Adapter*

*implements Phalcon\Image\AdapterInterface*

Image manipulation support. Allows images to be resized, cropped, etc.

**<?php**

```
$image = new Phalcon\Image\Adapter\GD("upload/test.jpg");
$image->resize(200, 200)->rotate(90)->crop(100, 100);
if ($image->save()) {
    echo 'success';
}
```

## Methods

public static boolean **check** ()

Checks if GD is enabled

public **\_\_construct** (*string* \$file, [*unknown* \$width], [*unknown* \$height])

Phalcon\Image\GD constructor

protected **\_resize** (*int* \$width, *int* \$height)

Execute a resize.

protected *Phalcon\Image\Adapter* **\_liquidRescale** (*unknown* \$width, *unknown* \$height, *unknown* \$delta\_x, *unknown* \$regidity)

This method scales the images using liquid rescaling method. Only support Imagick

protected **\_crop** (*int* \$width, *int* \$height, *int* \$offset\_x, *int* \$offset\_y)

Execute a crop.

protected **\_rotate** (*int* \$degrees)

Execute a rotation.

protected **\_flip** (*int* \$direction)

Execute a flip.

protected **\_sharpen** (*int* \$amount)

Execute a sharpen.

protected **\_reflection** (*int* \$height, *int* \$opacity, *boolean* \$fade\_in)

Execute a reflection.

protected **\_watermark** (*Phalcon\Image\Adapter* \$watermark, *int* \$offset\_x, *int* \$offset\_y, *int* \$opacity)

Execute a watermarking.

protected **\_text** (*unknown* \$text, *int* \$offset\_x, *int* \$offset\_y, *int* \$opacity, *int* \$r, *int* \$g, *int* \$b, *int* \$size, *string* \$fontfile)

Execute a text

protected **\_mask** (*unknown* \$mask)

Composite one image onto another

protected **\_background** (*int* \$r, *int* \$g, *int* \$b, *int* \$opacity)

Execute a background.

protected **\_blur** (*unknown* \$radius)

Blur image

protected **\_pixelate** (*unknown* \$amount)

Pixelate image

protected *boolean* **\_save** (*string* \$file, *int* \$quality)

Execute a save.

protected *string* **\_render** (*string* \$type, *int* \$quality)

Execute a render.

`protected resource __create (int $width, int $height)`

Create an empty image with the given width and height.

`public __destruct ()`

Destroys the loaded image to free up resources.

`public string getRealPath ()` inherited from Phalcon\Image\Adapter

Returns the real path of the image file

`public int getWidth ()` inherited from Phalcon\Image\Adapter

Returns the width of images

`public int getHeight ()` inherited from Phalcon\Image\Adapter

Returns the height of images

`public int getType ()` inherited from Phalcon\Image\Adapter

Returns the type of images

`public string getMime ()` inherited from Phalcon\Image\Adapter

Returns the mime of images

`public resource getImage ()` inherited from Phalcon\Image\Adapter

Returns the image of images

`public Phalcon\Image\Adapter resize ([unknown $width], [unknown $height], [unknown $master])` inherited from Phalcon\Image\Adapter

Resize the image to the given size. Either the width or the height can be omitted and the image will be resized proportionally.

`public Phalcon\Image\Adapter liquidRescale (unknown $width, unknown $height, [unknown $delta_x], [unknown $rigidity])` inherited from Phalcon\Image\Adapter

This method scales the images using liquid rescaling method. Only support Imagick

`public Phalcon\Image\Adapter crop (unknown $width, unknown $height, [unknown $offset_x], [unknown $offset_y])` inherited from Phalcon\Image\Adapter

Crop an image to the given size. Either the width or the height can be omitted and the current width or height will be used.

`public Phalcon\Image\Adapter rotate (unknown $degrees)` inherited from Phalcon\Image\Adapter

Rotate the image by a given amount.

`public Phalcon\Image\Adapter flip (unknown $direction)` inherited from Phalcon\Image\Adapter

Flip the image along the horizontal or vertical axis.

`public Phalcon\Image\Adapter sharpen (unknown $amount)` inherited from Phalcon\Image\Adapter

Sharpen the image by a given amount.

`public Phalcon\Image\Adapter reflection ([unknown $height], [unknown $opacity], [unknown $fade_in])` inherited from Phalcon\Image\Adapter

Add a reflection to an image. The most opaque part of the reflection will be equal to the opacity setting and fade out to full transparent. Alpha transparency is preserved.

public *Phalcon\Image\AdapterInterface* **watermark** (*unknown* \$watermark, [*unknown* \$offset\_x], [*unknown* \$offset\_y], [*unknown* \$opacity]) inherited from Phalcon\Image\Adapter

Add a watermark to an image with a specified opacity. Alpha transparency will be preserved.

public *Phalcon\Image\Adapter* **text** (*unknown* \$text, [*unknown* \$offset\_x], [*unknown* \$offset\_y], [*unknown* \$opacity], [*unknown* \$color], [*unknown* \$size], [*unknown* \$fontfile]) inherited from Phalcon\Image\Adapter

Add a text to an image with a specified opacity.

public *Phalcon\Image\Adapter* **mask** (*unknown* \$mask) inherited from Phalcon\Image\Adapter

Composite one image onto another

public *Phalcon\Image\Adapter* **background** (*unknown* \$color, [*unknown* \$quality]) inherited from Phalcon\Image\Adapter

Set the background color of an image. This is only useful for images with alpha transparency.

public *Phalcon\Image\Adapter* **blur** ([*unknown* \$radius]) inherited from Phalcon\Image\Adapter

Blur image

public *Phalcon\Image\Adapter* **pixelate** ([*unknown* \$amount]) inherited from Phalcon\Image\Adapter

Pixelate image

public *boolean* **save** ([*unknown* \$file], [*unknown* \$quality]) inherited from Phalcon\Image\Adapter

Save the image. If the filename is omitted, the original image will be overwritten.

public *Phalcon\Image\Adapter* **render** ([*unknown* \$type], [*unknown* \$quality]) inherited from Phalcon\Image\Adapter

Render the image and return the binary string.

## 2.54.125 Class Phalcon\Image\Adapter\Imagick

*extends* abstract class *Phalcon\Image\Adapter*

*implements* *Phalcon\Image\AdapterInterface*

Image manipulation support. Allows images to be resized, cropped, etc.

```
<?php
```

```
$image = new Phalcon\Image\Adapter\Imagick("upload/test.jpg");
$image->resize(200, 200)->rotate(90)->crop(100, 100);
if ($image->save()) {
    echo 'success';
}
```

### Methods

public static *boolean* **check** ()

Checks if Imagick is enabled

public **\_\_construct** (*string* \$file, [*unknown* \$width], [*unknown* \$height])

Phalcon\Image\Imagick constructor

protected **\_resize** (*int* \$width, *int* \$height)

Execute a resize.

**protected `_liquidRescale`** (*unknown \$width*, *unknown \$height*, *unknown \$delta\_x*, *unknown \$regidity*)

This method scales the images using liquid rescaling method. Only support Imagick

**protected `_crop`** (*int \$width*, *int \$height*, *int \$offset\_x*, *int \$offset\_y*)

Execute a crop.

**protected `_rotate`** (*int \$degrees*)

Execute a rotation.

**protected `_flip`** (*int \$direction*)

Execute a flip.

**protected `_sharpen`** (*int \$amount*)

Execute a sharpen.

**protected `_reflection`** (*int \$height*, *int \$opacity*, *boolean \$fade\_in*)

Execute a reflection.

**protected `_watermark`** (*Phalcon\Image\Adapter \$watermark*, *int \$offset\_x*, *int \$offset\_y*, *int \$opacity*)

Execute a watermarking.

**protected `_text`** (*unknown \$text*, *int \$offset\_x*, *int \$offset\_y*, *int \$opacity*, *int \$r*, *int \$g*, *int \$b*, *int \$size*, *string \$fontfile*)

Execute a text

**protected `_mask`** (*unknown \$mask*)

Composite one image onto another

**protected `_background`** (*int \$r*, *int \$g*, *int \$b*, *int \$opacity*)

Execute a background.

**protected `_blur`** (*unknown \$radius*)

Blur image

**protected `_pixelate`** (*unknown \$amount*)

Pixelate image

**protected boolean `_save`** (*string \$file*, *int \$quality*)

Execute a save.

**protected string `_render`** (*string \$type*, *int \$quality*)

Execute a render.

**public `__destruct`** ()

Destroys the loaded image to free up resources.

**public `getInternalImInstance`** ()

...

**public static `setResourceLimit`** (*unknown \$resource*, *unknown \$limit*)

...

public *string* **getRealPath** () inherited from Phalcon\Image\Adapter

Returns the real path of the image file

public *int* **getWidth** () inherited from Phalcon\Image\Adapter

Returns the width of images

public *int* **getHeight** () inherited from Phalcon\Image\Adapter

Returns the height of images

public *int* **getType** () inherited from Phalcon\Image\Adapter

Returns the type of images

public *string* **getMime** () inherited from Phalcon\Image\Adapter

Returns the mime of images

public *resource* **getImage** () inherited from Phalcon\Image\Adapter

Returns the image of images

public *Phalcon\Image\Adapter* **resize** ([*unknown \$width*], [*unknown \$height*], [*unknown \$master*]) inherited from Phalcon\Image\Adapter

Resize the image to the given size. Either the width or the height can be omitted and the image will be resized proportionally.

public *Phalcon\Image\Adapter* **liquidRescale** (*unknown \$width*, *unknown \$height*, [*unknown \$delta\_x*], [*unknown \$rigidity*]) inherited from Phalcon\Image\Adapter

This method scales the images using liquid rescaling method. Only support Imagick

public *Phalcon\Image\Adapter* **crop** (*unknown \$width*, *unknown \$height*, [*unknown \$offset\_x*], [*unknown \$offset\_y*]) inherited from Phalcon\Image\Adapter

Crop an image to the given size. Either the width or the height can be omitted and the current width or height will be used.

public *Phalcon\Image\Adapter* **rotate** (*unknown \$degrees*) inherited from Phalcon\Image\Adapter

Rotate the image by a given amount.

public *Phalcon\Image\Adapter* **flip** (*unknown \$direction*) inherited from Phalcon\Image\Adapter

Flip the image along the horizontal or vertical axis.

public *Phalcon\Image\Adapter* **sharpen** (*unknown \$amount*) inherited from Phalcon\Image\Adapter

Sharpen the image by a given amount.

public *Phalcon\Image\Adapter* **reflection** ([*unknown \$height*], [*unknown \$opacity*], [*unknown \$fade\_in*]) inherited from Phalcon\Image\Adapter

Add a reflection to an image. The most opaque part of the reflection will be equal to the opacity setting and fade out to full transparent. Alpha transparency is preserved.

public *Phalcon\Image\AdapterInterface* **watermark** (*unknown \$watermark*, [*unknown \$offset\_x*], [*unknown \$offset\_y*], [*unknown \$opacity*]) inherited from Phalcon\Image\Adapter

Add a watermark to an image with a specified opacity. Alpha transparency will be preserved.

public *Phalcon\Image\Adapter* **text** (*unknown \$text*, [*unknown \$offset\_x*], [*unknown \$offset\_y*], [*unknown \$opacity*], [*unknown \$color*], [*unknown \$size*], [*unknown \$fontfile*]) inherited from Phalcon\Image\Adapter

Add a text to an image with a specified opacity.

public *Phalcon\Image\Adapter* **mask** (*unknown* \$mask) inherited from *Phalcon\Image\Adapter*

Composite one image onto another

public *Phalcon\Image\Adapter* **background** (*unknown* \$color, [*unknown* \$quality]) inherited from *Phalcon\Image\Adapter*

Set the background color of an image. This is only useful for images with alpha transparency.

public *Phalcon\Image\Adapter* **blur** ([*unknown* \$radius]) inherited from *Phalcon\Image\Adapter*

Blur image

public *Phalcon\Image\Adapter* **pixelate** ([*unknown* \$amount]) inherited from *Phalcon\Image\Adapter*

Pixelate image

public *boolean* **save** ([*unknown* \$file], [*unknown* \$quality]) inherited from *Phalcon\Image\Adapter*

Save the image. If the filename is omitted, the original image will be overwritten.

public *Phalcon\Image\Adapter* **render** ([*unknown* \$type], [*unknown* \$quality]) inherited from *Phalcon\Image\Adapter*

Render the image and return the binary string.

## 2.54.126 Class *Phalcon\Image\Exception*

*extends class Phalcon\Exception*

Exceptions thrown in *Phalcon\Image* will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from *Exception*

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from *Exception*

Exception constructor

final public *string* **getMessage** () inherited from *Exception*

Gets the Exception message

final public *int* **getCode** () inherited from *Exception*

Gets the Exception code

final public *string* **getFile** () inherited from *Exception*

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from *Exception*

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from *Exception*

Gets the stack trace

final public *Exception* **getPrevious** () inherited from *Exception*

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from *Exception*  
 Gets the stack trace as a string  
 public *string* **\_\_toString** () inherited from *Exception*  
 String representation of the exception

## 2.54.127 Class Phalcon\Kernel

### Methods

```
public static preComputeHashKey (unknown $arrKey)
...
public static preComputeHashKey32 (unknown $arrKey)
...
public static preComputeHashKey64 (unknown $arrKey)
...
```

## 2.54.128 Class Phalcon\Loader

*implements Phalcon\Events\EventsAwareInterface*

This component helps to load your project classes automatically based on some conventions

```
<?php
//Creates the autoloader
$loader = new Phalcon\Loader();

//Register some namespaces
$loader->registerNamespaces(array(
    'Example\Base' => 'vendor/example/base/',
    'Example\Adapter' => 'vendor/example/adapter/',
    'Example' => 'vendor/example/',
));
//register autoloader
$loader->register();

//Requiring this class will automatically include file vendor/example/adapter/Some.php
$adapter = Example\Adapter\Some();
```

### Methods

```
public __construct ()
Phalcon\Loader constructor
public setEventsManager (Phalcon\Events\ManagerInterface $eventsManager)
Sets the events manager
public Phalcon\Events\ManagerInterface getEventsManager ()
```

Returns the internal event manager

`public Phalcon\Loader setExtensions (array $extensions)`

Sets an array of extensions that the loader must try in each attempt to locate the file

`public boolean getExtensions ()`

Return file extensions registered in the loader

`public Phalcon\Loader registerNamespaces (array $namespaces, [boolean $merge])`

Register namespaces and their related directories

`public array getNamespaces ()`

Return current namespaces registered in the autoloader

`public Phalcon\Loader registerPrefixes (array $prefixes, [boolean $merge])`

Register directories on which “not found” classes could be found

`public getPrefixes ()`

Return current prefixes registered in the autoloader

`public Phalcon\Loader registerDirs (array $directories, [boolean $merge])`

Register directories on which “not found” classes could be found

`public getDirs ()`

Return current directories registered in the autoloader

`public Phalcon\Loader registerClasses (array $classes, [boolean $merge])`

Register classes and their locations

`public getClasses ()`

Return the current class-map registered in the autoloader

`public Phalcon\Loader register ()`

Register the autoload method

`public Phalcon\Loader unregister ()`

Unregister the autoload method

`public boolean autoLoad (string $className)`

Makes the work of autoload registered classes

`public string getFoundPath ()`

Get the path when a class was found

`public string getCheckedPath ()`

Get the path the loader is checking for a path

## 2.54.129 Class Phalcon\Loader\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Loader will use this class

## Methods

final private *Exception* **`__clone()`** inherited from *Exception*  
Clone the exception

public **`__construct ([string $message], [int $code], [Exception $previous])`** inherited from *Exception*  
*Exception* constructor

final public *string* **`getMessage ()`** inherited from *Exception*  
Gets the *Exception* message

final public *int* **`getCode ()`** inherited from *Exception*  
Gets the *Exception* code

final public *string* **`getFile ()`** inherited from *Exception*  
Gets the file in which the exception occurred

final public *int* **`getLine ()`** inherited from *Exception*  
Gets the line in which the exception occurred

final public *array* **`getTrace ()`** inherited from *Exception*  
Gets the stack trace

final public *Exception* **`getPrevious ()`** inherited from *Exception*  
Returns previous *Exception*

final public *Exception* **`getTraceAsString ()`** inherited from *Exception*  
Gets the stack trace as a string

public *string* **`__toString ()`** inherited from *Exception*  
String representation of the exception

## 2.54.130 Abstract class Phalcon\Logger

*Phalcon\Logger* is a component whose purpose is create logs using different backends via adapters, generating options, formats and filters also implementing transactions.

```
<?php

$logger = new Phalcon\Logger\Adapter\File("app/logs/test.log");
$logger->log("This is a message");
$logger->log("This is an error", Phalcon\Logger::ERROR);
$logger->error("This is another error");
```

## Constants

*integer* **SPECIAL**  
*integer* **CUSTOM**  
*integer* **DEBUG**  
*integer* **INFO**

*integer NOTICE*  
*integer WARNING*  
*integer ERROR*  
*integer ALERT*  
*integer CRITICAL*  
*integer EMERGENCE*  
*integer EMERGENCY*

### 2.54.131 Abstract class Phalcon\Logger\Adapter

*implements Phalcon\Logger\AdapterInterface*

Base class for Phalcon\Logger adapters

#### Methods

**public *Phalcon\Logger\Adapter* setLogLevel (*int \$level*)**

Filters the logs sent to the handlers that are less or equal than a specific level

**public *int getLogLevel ()***

Returns the current log level

**public *Phalcon\Logger\Adapter setFormatter (*Phalcon\Logger\FormatterInterface \$formatter*)***

Sets the message formatter

**public *Phalcon\Logger\Adapter isTransaction ()***

Returns the current transaction

**public *Phalcon\Logger\Adapter begin ()***

Starts a transaction

**public *Phalcon\Logger\Adapter commit ()***

Commits the internal transaction

**public *Phalcon\Logger\Adapter rollback ()***

Rollbacks the internal transaction

**public *emergence (*unknown \$message, [unknown \$context]*)***

...

**public *Phalcon\Logger\Adapter log (*unknown \$type, string \$message, [array \$context]*)***

Logs messages to the internal logger. Appends messages to the log

**public *Phalcon\Logger\AdapterInterface debug (*string \$message, [array \$context]*)***

Sends/Writes a debug message to the log

**public *Phalcon\Logger\AdapterInterface info (*string \$message, [array \$context]*)***

Sends/Writes an info message to the log

**public *Phalcon\Logger\AdapterInterface notice (*string \$message, [array \$context]*)***

Sends/Writes a notice message to the log

```
public Phalcon\Logger\AdapterInterface warning (string $message, [array $context])
```

Sends/Writes a warning message to the log

```
public Phalcon\Logger\AdapterInterface error (string $message, [array $context])
```

Sends/Writes an error message to the log

```
public Phalcon\Logger\AdapterInterface critical (string $message, [array $context])
```

Sends/Writes a critical message to the log

```
public Phalcon\Logger\AdapterInterface alert (string $message, [array $context])
```

Sends/Writes an alert message to the log

```
public Phalcon\Logger\AdapterInterface emergency (string $message, [array $context])
```

Sends/Writes an emergency message to the log

```
abstract protected logInternal (unknown $message, unknown $type, unknown $time, unknown $context)
```

...

```
abstract public Phalcon\Logger\FormatterInterface getFormatter () inherited from Phalcon\Logger\AdapterInterface
```

Returns the internal formatter

```
abstract public boolean close () inherited from Phalcon\Logger\AdapterInterface
```

Closes the logger

### 2.54.132 Class Phalcon\Logger\Adapter\File

*extends abstract class Phalcon\Logger\Adapter*

*implements Phalcon\Logger\AdapterInterface*

Adapter to store logs in plain text files

<?php

```
$logger = new \Phalcon\Logger\Adapter\File("app/logs/test.log");
$logger->log("This is a message");
$logger->log("This is an error", \Phalcon\Logger::ERROR);
$logger->error("This is another error");
$logger->close();
```

#### Methods

```
public __construct (string $name, [array $options])
```

Phalcon\Logger\Adapter\File constructor

```
public Phalcon\Logger\Formatter\Line getFormatter ()
```

Returns the internal formatter

```
protected logInternal (string $message, int $type, int $time, array $context)
```

Writes the log to the file itself

public **boolean close ()**

Closes the logger

public **getPath ()**

Returns the file path

public **\_\_wakeup ()**

Opens the internal file handler after unserialization

public *Phalcon\Logger\Adapter* **setLogLevel (int \$level)** inherited from *Phalcon\Logger\Adapter*

Filters the logs sent to the handlers that are less or equal than a specific level

public **int getLogLevel ()** inherited from *Phalcon\Logger\Adapter*

Returns the current log level

public *Phalcon\Logger\Adapter* **setFormatter (*Phalcon\Logger\FormatterInterface* \$formatter)** inherited from *Phalcon\Logger\Adapter*

Sets the message formatter

public *Phalcon\Logger\Adapter* **isTransaction ()** inherited from *Phalcon\Logger\Adapter*

Returns the current transaction

public *Phalcon\Logger\Adapter* **begin ()** inherited from *Phalcon\Logger\Adapter*

Starts a transaction

public *Phalcon\Logger\Adapter* **commit ()** inherited from *Phalcon\Logger\Adapter*

Commits the internal transaction

public *Phalcon\Logger\Adapter* **rollback ()** inherited from *Phalcon\Logger\Adapter*

Rollbacks the internal transaction

public **emergence (unknown \$message, [unknown \$context])** inherited from *Phalcon\Logger\Adapter*

...

public *Phalcon\Logger\Adapter* **log (unknown \$type, string \$message, [array \$context])** inherited from *Phalcon\Logger\Adapter*

Logs messages to the internal logger. Appends messages to the log

public *Phalcon\Logger\AdapterInterface* **debug (string \$message, [array \$context])** inherited from *Phalcon\Logger\Adapter*

Sends/Writes a debug message to the log

public *Phalcon\Logger\AdapterInterface* **info (string \$message, [array \$context])** inherited from *Phalcon\Logger\Adapter*

Sends/Writes an info message to the log

public *Phalcon\Logger\AdapterInterface* **notice (string \$message, [array \$context])** inherited from *Phalcon\Logger\Adapter*

Sends/Writes a notice message to the log

public *Phalcon\Logger\AdapterInterface* **warning (string \$message, [array \$context])** inherited from *Phalcon\Logger\Adapter*

Sends/Writes a warning message to the log

`public Phalcon\Logger\AdapterInterface error (string $message, [array $context])` inherited from `Phalcon\Logger\Adapter`

Sends/Writes an error message to the log

`public Phalcon\Logger\AdapterInterface critical (string $message, [array $context])` inherited from `Phalcon\Logger\Adapter`

Sends/Writes a critical message to the log

`public Phalcon\Logger\AdapterInterface alert (string $message, [array $context])` inherited from `Phalcon\Logger\Adapter`

Sends/Writes an alert message to the log

`public Phalcon\Logger\AdapterInterface emergency (string $message, [array $context])` inherited from `Phalcon\Logger\Adapter`

Sends/Writes an emergency message to the log

### 2.54.133 Class `Phalcon\Logger\Adapter\Firephp`

*extends abstract class `Phalcon\Logger\Adapter`*

*implements `Phalcon\Logger\AdapterInterface`*

Sends logs to FirePHP

`<?php`

```
$logger = new \Phalcon\Logger\Adapter\Firephp("");
$logger->log("This is a message");
$logger->log("This is an error", \Phalcon\Logger::ERROR);
$logger->error("This is another error");
```

#### Methods

`public Phalcon\Logger\FormatterInterface getFormatter ()`

Returns the internal formatter

`protected logInternal (string $message, int $type, int $time, array $context)`

Writes the log to the stream itself

`public boolean close ()`

Closes the logger

`public Phalcon\Logger\Adapter setLogLevel (int $level)` inherited from `Phalcon\Logger\Adapter`

Filters the logs sent to the handlers that are less or equal than a specific level

`public int getLogLevel ()` inherited from `Phalcon\Logger\Adapter`

Returns the current log level

`public Phalcon\Logger\Adapter setFormatter (Phalcon\Logger\FormatterInterface $formatter)` inherited from `Phalcon\Logger\Adapter`

Sets the message formatter

`public Phalcon\Logger\Adapter isTransaction ()` inherited from `Phalcon\Logger\Adapter`

Returns the current transaction

public *Phalcon\Logger\Adapter* **begin** () inherited from *Phalcon\Logger\Adapter*

Starts a transaction

public *Phalcon\Logger\Adapter* **commit** () inherited from *Phalcon\Logger\Adapter*

Commits the internal transaction

public *Phalcon\Logger\Adapter* **rollback** () inherited from *Phalcon\Logger\Adapter*

Rollbacks the internal transaction

public **emergence** (*unknown* \$message, [*unknown* \$context]) inherited from *Phalcon\Logger\Adapter*

...

public *Phalcon\Logger\Adapter* **log** (*unknown* \$type, *string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Logs messages to the internal logger. Appends messages to the log

public *Phalcon\Logger\AdapterInterface* **debug** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a debug message to the log

public *Phalcon\Logger\AdapterInterface* **info** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an info message to the log

public *Phalcon\Logger\AdapterInterface* **notice** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a notice message to the log

public *Phalcon\Logger\AdapterInterface* **warning** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a warning message to the log

public *Phalcon\Logger\AdapterInterface* **error** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an error message to the log

public *Phalcon\Logger\AdapterInterface* **critical** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a critical message to the log

public *Phalcon\Logger\AdapterInterface* **alert** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an alert message to the log

public *Phalcon\Logger\AdapterInterface* **emergency** (*string* \$message, [*array* \$context]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an emergency message to the log

## 2.54.134 Class Phalcon\Logger\Adapter\Stream

*extends abstract class Phalcon\Logger\Adapter*

*implements Phalcon\Logger\AdapterInterface*

Sends logs to a valid PHP stream

<?php

```
$logger = new \Phalcon\Logger\Adapter\Stream("php://stderr");
$logger->log("This is a message");
$logger->log("This is an error", \Phalcon\Logger::ERROR);
$logger->error("This is another error");
```

### Methods

public **\_\_construct** (*string \$name, [array \$options]*)

Phalcon\Logger\Adapter\Stream constructor

public *Phalcon\Logger\Formatter\Line getFormatter ()*

Returns the internal formatter

protected **logInternal** (*string \$message, int \$type, int \$time, array \$context*)

Writes the log to the stream itself

public *boolean close ()*

Closes the logger

public *Phalcon\Logger\Adapter setLogLevel (int \$level)* inherited from Phalcon\Logger\Adapter

Filters the logs sent to the handlers that are less or equal than a specific level

public *int getLogLevel ()* inherited from Phalcon\Logger\Adapter

Returns the current log level

public *Phalcon\Logger\Adapter setFormatter (Phalcon\Logger\FormatterInterface \$formatter)* inherited from Phalcon\Logger\Adapter

Sets the message formatter

public *Phalcon\Logger\Adapter isTransaction ()* inherited from Phalcon\Logger\Adapter

Returns the current transaction

public *Phalcon\Logger\Adapter begin ()* inherited from Phalcon\Logger\Adapter

Starts a transaction

public *Phalcon\Logger\Adapter commit ()* inherited from Phalcon\Logger\Adapter

Commits the internal transaction

public *Phalcon\Logger\Adapter rollback ()* inherited from Phalcon\Logger\Adapter

Rollbacks the internal transaction

public **emergence** (*unknown \$message, [unknown \$context]*) inherited from Phalcon\Logger\Adapter

...

public *Phalcon\Logger\Adapter* **log** (*unknown \$type*, *string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Logs messages to the internal logger. Appends messages to the log

public *Phalcon\Logger\AdapterInterface* **debug** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a debug message to the log

public *Phalcon\Logger\AdapterInterface* **info** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an info message to the log

public *Phalcon\Logger\AdapterInterface* **notice** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a notice message to the log

public *Phalcon\Logger\AdapterInterface* **warning** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a warning message to the log

public *Phalcon\Logger\AdapterInterface* **error** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an error message to the log

public *Phalcon\Logger\AdapterInterface* **critical** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes a critical message to the log

public *Phalcon\Logger\AdapterInterface* **alert** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an alert message to the log

public *Phalcon\Logger\AdapterInterface* **emergency** (*string \$message*, [*array \$context*]) inherited from *Phalcon\Logger\Adapter*

Sends/Writes an emergency message to the log

## 2.54.135 Class *Phalcon\Logger\Adapter\Syslog*

*extends* abstract class *Phalcon\Logger\Adapter*

*implements* *Phalcon\Logger\AdapterInterface*

Sends logs to the system logger

```
<?php
```

```
$logger = new \Phalcon\Logger\Adapter\Syslog("ident", array(
    'option' => LOG_NDELAY,
    'facility' => LOG_MAIL
));
$logger->log("This is a message");
$logger->log("This is an error", \Phalcon\Logger::ERROR);
$logger->error("This is another error");
```

## Methods

`public __construct (string $name, [array $options])`

Phalcon\Logger\Adapter\Syslog constructor

`public Phalcon\Logger\Formatter\Line getFormatter ()`

Returns the internal formatter

`protected logInternal (string $message, int $type, int $time, array $context)`

Writes the log to the stream itself

`public boolean close ()`

Closes the logger

`public Phalcon\Logger\Adapter setLogLevel (int $level)` inherited from Phalcon\Logger\Adapter

Filters the logs sent to the handlers that are less or equal than a specific level

`public int getLogLevel ()` inherited from Phalcon\Logger\Adapter

Returns the current log level

`public Phalcon\Logger\Adapter setFormatter (Phalcon\Logger\FormatterInterface $formatter)` inherited from Phalcon\Logger\Adapter

Sets the message formatter

`public Phalcon\Logger\Adapter isTransaction ()` inherited from Phalcon\Logger\Adapter

Returns the current transaction

`public Phalcon\Logger\Adapter begin ()` inherited from Phalcon\Logger\Adapter

Starts a transaction

`public Phalcon\Logger\Adapter commit ()` inherited from Phalcon\Logger\Adapter

Commits the internal transaction

`public Phalcon\Logger\Adapter rollback ()` inherited from Phalcon\Logger\Adapter

Rollbacks the internal transaction

`public emergence (unknown $message, [unknown $context])` inherited from Phalcon\Logger\Adapter

...

`public Phalcon\Logger\Adapter log (unknown $type, string $message, [array $context])` inherited from Phalcon\Logger\Adapter

Logs messages to the internal logger. Appends messages to the log

`public Phalcon\Logger\AdapterInterface debug (string $message, [array $context])` inherited from Phalcon\Logger\Adapter

Sends/Writes a debug message to the log

`public Phalcon\Logger\AdapterInterface info (string $message, [array $context])` inherited from Phalcon\Logger\Adapter

Sends/Writes an info message to the log

`public Phalcon\Logger\AdapterInterface notice (string $message, [array $context])` inherited from Phalcon\Logger\Adapter

Sends/Writes a notice message to the log

```
public Phalcon\Logger\AdapterInterface warning (string $message, [array $context]) inherited from  
Phalcon\Logger\Adapter
```

Sends/Writes a warning message to the log

```
public Phalcon\Logger\AdapterInterface error (string $message, [array $context]) inherited from  
Phalcon\Logger\Adapter
```

Sends/Writes an error message to the log

```
public Phalcon\Logger\AdapterInterface critical (string $message, [array $context]) inherited from  
Phalcon\Logger\Adapter
```

Sends/Writes a critical message to the log

```
public Phalcon\Logger\AdapterInterface alert (string $message, [array $context]) inherited from  
Phalcon\Logger\Adapter
```

Sends/Writes an alert message to the log

```
public Phalcon\Logger\AdapterInterface emergency (string $message, [array $context]) inherited from  
Phalcon\Logger\Adapter
```

Sends/Writes an emergency message to the log

## 2.54.136 Class Phalcon\Logger\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Logger will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

```
public __construct ([string $message], [int $code], [Exception $previous]) inherited from Exception
```

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

```
final public int getLine () inherited from Exception
```

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

`final public Exception getTraceAsString ()` inherited from `Exception`

Gets the stack trace as a string

`public string __toString ()` inherited from `Exception`

String representation of the exception

### 2.54.137 Abstract class Phalcon\Logger\Formatter

*implements Phalcon\Logger\FormatterInterface*

This is a base class for logger formatters

#### Methods

`public string getTypeString (integer $type)`

Returns the string meaning of a logger constant

`protected interpolate (string $message, array $context)`

Interpolates context values into the message placeholders

`abstract public format (string $message, int $type, int $timestamp, array $context)` inherited from `Phalcon\Logger\FormatterInterface`

Applies a format to a message before sent it to the internal log

### 2.54.138 Class Phalcon\Logger\Formatter\Firephp

*extends abstract class Phalcon\Logger\Formatter*

*implements Phalcon\Logger\FormatterInterface*

Formats messages so that they can be sent to FirePHP

#### Methods

`public string getTypeString (integer $type)`

Returns the string meaning of a logger constant

`public getShowBacktrace ()`

...

`public setShowBacktrace ([unknown $show])`

...

`public enableLabels ([unknown $enable])`

...

`public labelsEnabled ()`

...

`public string format (string $message, int $type, int $timestamp, array $context)`

Applies a format to a message before sending it to the log

protected **interpolate** (*string \$message, array \$context*) inherited from Phalcon\Logger\Formatter

Interpolates context values into the message placeholders

### 2.54.139 Class Phalcon\Logger\Formatter\Json

*extends abstract class Phalcon\Logger\Formatter*

*implements Phalcon\Logger\FormatterInterface*

Formats messages using JSON encoding

#### Methods

public **format** (*string \$message, int \$type, int \$timestamp, array \$context*)

Applies a format to a message before sent it to the internal log

public **getTypeString** (*integer \$type*) inherited from Phalcon\Logger\Formatter

Returns the string meaning of a logger constant

protected **interpolate** (*string \$message, array \$context*) inherited from Phalcon\Logger\Formatter

Interpolates context values into the message placeholders

### 2.54.140 Class Phalcon\Logger\Formatter\Line

*extends abstract class Phalcon\Logger\Formatter*

*implements Phalcon\Logger\FormatterInterface*

Formats messages using an one-line string

#### Methods

public **\_\_construct** ([*string \$format, [string \$dateFormat]*])

Phalcon\Logger\Formatter\Line construct

public **setFormat** (*string \$format*)

Set the log format

public **format getFormat** ()

Returns the log format

public **setDateFormat** (*string \$date*)

Sets the internal date format

public **string getDateFormat** ()

Returns the internal date format

public **string format** (*string \$message, int \$type, int \$timestamp, array \$context*)

Applies a format to a message before sent it to the internal log

public *string* **getTypeString** (*integer* \$type) inherited from Phalcon\Logger\Formatter

Returns the string meaning of a logger constant

protected  **interpolate** (*string* \$message, *array* \$context) inherited from Phalcon\Logger\Formatter

Interpolates context values into the message placeholders

### 2.54.141 Class Phalcon\Logger\Formatter\Syslog

*extends* abstract class Phalcon\Logger\Formatter

*implements* Phalcon\Logger\FormatterInterface

Prepares a message to be used in a Syslog backend

#### Methods

public *array* **format** (*string* \$message, *int* \$type, *int* \$timestamp, *array* \$context)

Applies a format to a message before sent it to the internal log

public *string* **getTypeString** (*integer* \$type) inherited from Phalcon\Logger\Formatter

Returns the string meaning of a logger constant

protected  **interpolate** (*string* \$message, *array* \$context) inherited from Phalcon\Logger\Formatter

Interpolates context values into the message placeholders

### 2.54.142 Class Phalcon\Logger\Item

Represents each item in a logging transaction

#### Methods

public **\_\_construct** (*string* \$message, *integer* \$type, [*integer* \$time])

Phalcon\Logger\Item constructor

public *string* **getMessage** ()

Returns the message

public *integer* **getType** ()

Returns the log type

public *integer* **getTime** ()

Returns log timestamp

public **getContext** ()

...

### 2.54.143 Class Phalcon\Logger\Multiple

Handles multiples logger handlers

## Methods

public **push** (*Phalcon\Logger\AdapterInterface* \$logger)

Pushes a logger to the logger tail

public *Phalcon\Logger\AdapterInterface* [] **getLoggers** ()

Returns the registered loggers

public **setFormatter** (*Phalcon\Logger\FormatterInterface* \$formatter)

Sets a global formatter

public *Phalcon\Logger\FormatterInterface* **getFormatter** ()

Returns a formatter

public **log** (*int* \$type, *string* \$message, [*unknown* \$context])

Sends a message to each registered logger

public **emergency** (*string* \$message, [*unknown* \$context])

Sends/Writes an emergency message to the log

public **emergence** (*unknown* \$message, [*unknown* \$context])

...

public **debug** (*string* \$message, [*unknown* \$context])

Sends/Writes a debug message to the log

public **error** (*string* \$message, [*unknown* \$context])

Sends/Writes an error message to the log

public **info** (*string* \$message, [*unknown* \$context])

Sends/Writes an info message to the log

public **notice** (*string* \$message, [*unknown* \$context])

Sends/Writes a notice message to the log

public **warning** (*string* \$message, [*unknown* \$context])

Sends/Writes a warning message to the log

public **alert** (*string* \$message, [*unknown* \$context])

Sends/Writes an alert message to the log

## 2.54.144 Class Phalcon\Mvc\Application

*extends abstract class Phalcon\DI\Injectable*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\DI\InjectionAwareInterface*

This component encapsulates all the complex operations behind instantiating every component needed and integrating it with the rest to allow the MVC pattern to operate as desired.

```
<?php

class Application extends \Phalcon\Mvc\Application
{

    /**
     * Register the services here to make them general or register
     * in the ModuleDefinition to make them module-specific
     */
    protected function _registerServices()
    {

    }

    /**
     * This method registers all the modules in the application
     */
    public function main()
    {
        $this->registerModules(array(
            'frontend' => array(
                'className' => 'Multiple\Frontend\Module',
                'path' => '../apps/frontend/Module.php'
            ),
            'backend' => array(
                'className' => 'Multiple\Backend\Module',
                'path' => '../apps/backend/Module.php'
            )
        ));
    }
}

$application = new Application();
$application->main();
```

## Methods

```
public __construct ([Phalcon\DI $dependencyInjector])
public Phalcon\Mvc\Application useImplicitView (boolean $implicitView)
```

By default. The view is implicitly buffering all the output You can full disable the view component using this method

```
public registerModules (array $modules, [boolean $merge])
```

Register an array of modules present in the application

```
<?php

$this->registerModules(array(
    'frontend' => array(
        'className' => 'Multiple\Frontend\Module',
        'path' => '../apps/frontend/Module.php'
    ),
    'backend' => array(
        'className' => 'Multiple\Backend\Module',
        'path' => '../apps/backend/Module.php'
```

```
)  
));  
  
public array getModules ()  
Return the modules registered in the application  
  
public Phalcon\ Mvc\ Application setDefaultModule (string $defaultModule)  
Sets the module name to be used if the router doesn't return a valid module  
  
public string getDefaultModule ()  
Returns the default module name  
  
public Phalcon\ Http\ ResponseInterface handle ([string $uri])  
Handles a MVC request  
  
public setDI (Phalcon\ DiInterface $dependencyInjector) inherited from Phalcon\ DI\ Injectable  
Sets the dependency injector  
  
public Phalcon\ DiInterface getDI () inherited from Phalcon\ DI\ Injectable  
Returns the internal dependency injector  
  
public setEventsManager (Phalcon\ Events\ ManagerInterface $eventsManager) inherited from  
Phalcon\ DI\ Injectable  
Sets the event manager  
  
public Phalcon\ Events\ ManagerInterface getEventsManager () inherited from Phalcon\ DI\ Injectable  
Returns the internal event manager  
  
public __get (unknown $property) inherited from Phalcon\ DI\ Injectable  
Magic method __get
```

## 2.54.145 Class Phalcon\ Mvc\ Application\ Exception

extends class Phalcon\ Exception

Exceptions thrown in Phalcon\ Mvc\ Application class will use this class

### Methods

```
final private Exception __clone () inherited from Exception  
Clone the exception  
  
public __construct ([string $message], [int $code], [Exception $previous]) inherited from Exception  
Exception constructor  
  
final public string getMessage () inherited from Exception  
Gets the Exception message  
  
final public int getCode () inherited from Exception  
Gets the Exception code  
  
final public string getFile () inherited from Exception
```

Gets the file in which the exception occurred  
final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred  
final public *array* **getTrace** () inherited from Exception

Gets the stack trace  
final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception  
final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string  
public *string* **\_\_toString** () inherited from Exception

String representation of the exception

## 2.54.146 Class Phalcon\Mvc\Collection

*implements* *Phalcon\ Mvc\ CollectionInterface*, *Phalcon\ DI\ InjectionAwareInterface*, *Serializable*

This component implements a high level abstraction for NoSQL databases which works with documents

### Constants

*integer* **OP\_NONE**

*integer* **OP\_CREATE**

*integer* **OP\_UPDATE**

*integer* **OP\_DELETE**

### Methods

final public **\_\_construct** ([*Phalcon\ DiInterface* \$dependencyInjector])

Phalcon\Mvc\Model constructor

public **setId** (*mixed* \$id)

Sets a value for the `_id` property, creates a `MongoId` object if needed

public *MongoId* **getId** ()

Returns the value of the `_id` property

public **setDI** (*Phalcon\ DiInterface* \$dependencyInjector)

Sets the dependency injection container

public *Phalcon\ DiInterface* **getDI** ()

Returns the dependency injection container

protected **setEventsManager** (*Phalcon\ Events\ ManagerInterface* \$eventsManager)

Sets a custom events manager

protected *Phalcon\ Events\ ManagerInterface* **getEventsManager** ()

Returns the custom events manager

```
public Phalcon\ Mvc\ Model\ ManagerInterface getModelsManager ()
```

Returns the models manager related to the entity instance

```
public array getReservedAttributes ()
```

Returns an array with reserved properties that cannot be part of the insert/update

```
protected useImplicitObjectIds ()
```

Sets if a model must use implicit objects ids

```
protected Phalcon\ Mvc\ Collection setSource ()
```

Sets collection name which model should be mapped

```
public string getSource ()
```

Returns collection name mapped in the model

```
public Phalcon\ Mvc\ Model setConnectionService (string $connectionService)
```

Sets the DependencyInjection connection service name

```
public string getConnectionService ()
```

Returns DependencyInjection connection service

```
public MongoDb getConnection ()
```

Retrieves a database connection

```
public mixed readAttribute (string $attribute)
```

Reads an attribute value by its name

```
<?php
```

```
echo $robot->readAttribute('name');
```

```
public writeAttribute (string $attribute, mixed $value)
```

Writes an attribute value by its name

```
<?php
```

```
$robot->writeAttribute('name', 'Rosey');
```

```
public static Phalcon\ Mvc\ Collection cloneResult (Phalcon\ Mvc\ Collection $collection, array $document)
```

Returns a cloned collection

```
protected static array _getResultSet ()
```

Returns a collection resultset

```
protected static int _getGroupResultSet ()
```

Perform a count over a resultset

```
protected boolean _preSave ()
```

Executes internal hooks before save a document

```
protected boolean _postSave ()
```

Executes internal events after save a document

protected **validate** ()

Executes validators on every validation call

<?php

```
use Phalcon\Mvc\Model\Validator\ExclusionIn as ExclusionIn;

class Subscriptors extends Phalcon\Mvc\Collection
{

    public function validation()
    {
        $this->validate(new ExclusionIn(array(
            'field' => 'status',
            'domain' => array('A', 'I')
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

public *boolean* validationHasFailed ()

Check whether validation process has generated any messages

<?php

```
use Phalcon\Mvc\Model\Validator\ExclusionIn as ExclusionIn;

class Subscriptors extends Phalcon\Mvc\Collection
{

    public function validation()
    {
        $this->validate(new ExclusionIn(array(
            'field' => 'status',
            'domain' => array('A', 'I')
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

public *boolean* fireEvent (*string* \$eventName)

Fires an internal event

public *boolean* fireEventCancel (*string* \$eventName)

Fires an internal event that cancels the operation

protected *boolean* \_cancelOperation ()

Cancel the current operation

protected *boolean* \_exists ()

Checks if the document exists in the collection

public *Phalcon\ Mvc\ Model\ MessageInterface* [] **getMessages** ()

Returns all the validation messages

<?php

```
$robot = new Robots();
$robot->type = 'mechanical';
$robot->name = 'Astro Boy';
$robot->year = 1952;
if ($robot->save() == false) {
    echo "Umh, We can't store robots right now ";
    foreach ($robot->getMessages() as $message) {
        echo $message;
    }
} else {
    echo "Great, a new robot was saved successfully!";
}
```

public **appendMessage** (*Phalcon\ Mvc\ Model\ MessageInterface* \$message)

Appends a customized message on the validation process

<?php

```
use \Phalcon\ Mvc\ Model\ Message as Message;

class Robots extends Phalcon\ Mvc\ Model
{

    public function beforeSave()
    {
        if ($this->name == 'Peter') {
            $message = new Message("Sorry, but a robot cannot be named Peter");
            $this->appendMessage($message);
        }
    }
}
```

public *boolean* **save** ()

Creates/Updates a collection based on the values in the attributes

public static *Phalcon\ Mvc\ Collection* **findById** (*string/MongoId* \$id)

Find a document by its id (\_id)

public static *array* **findFirst** ([*array* \$parameters])

Allows to query the first record that match the specified conditions

<?php

```
//What's the first robot in the robots table?
$robot = Robots::findFirst();
echo "The robot name is ", $robot->name, "\n";

//What's the first mechanical robot in robots table?
$robot = Robots::findFirst(array(
    array("type" => "mechanical")
));
echo "The first mechanical robot name is ", $robot->name, "\n";
```

```
//Get first virtual robot ordered by name
$robot = Robots::findFirst(array(
    array("type" => "mechanical"),
    "order" => array("name" => 1)
));
echo "The first virtual robot name is ", $robot->name, "\n";
```

public static *array* **find** ([*array* \$parameters])

Allows to query a set of records that match the specified conditions

```
<?php
```

```
//How many robots are there?
$robots = Robots::find();
echo "There are ", count($robots), "\n";

//How many mechanical robots are there?
$robots = Robots::find(array(
    array("type" => "mechanical")
));
echo "There are ", count($robots), "\n";

//Get and print virtual robots ordered by name
$robots = Robots::findFirst(array(
    array("type" => "virtual"),
    "order" => array("name" => 1)
));
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

//Get first 100 virtual robots ordered by name
$robots = Robots::find(array(
    array("type" => "virtual"),
    "order" => array("name" => 1),
    "limit" => 100
));
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}
```

public static *array* **count** ([*array* \$parameters])

Perform a count over a collection

```
<?php
```

```
echo 'There are ', Robots::count(), ' robots';
```

public static *array* **aggregate** (*array* \$parameters)

Perform an aggregation using the Mongo aggregation framework

public static *array* **summatory** (*string* \$field, [*array* \$conditions], [*string* \$finalize])

Allows to perform a summatory group for a column in the collection

public *boolean* **delete** ()

Deletes a model instance. Returning true on success or false otherwise.

<?php

```
$robot = Robots::findFirst();
$robot->delete();

foreach (Robots::find() as $robot) {
    $robot->delete();
}
```

public *array* **toArray** ()

Returns the instance as an array representation

<?php

```
print_r($robot->toArray());
```

public *string* **serialize** ()

Serializes the object ignoring connections or protected properties

public **unserialize** ([*unknown* \$serialized])

Unserializes the object from a serialized string

public static *array* **execute** (*mixed* \$code, [*array* \$args])

Runs JavaScript code on the database server.

<?php

```
$ret = Robots::execute("function() { return 'Hello, world!';}");
echo $ret['retval'], "\n";
```

## 2.54.147 Class Phalcon\Mvc\Collection\Document

*implements* ArrayAccess

This component allows Phalcon\Mvc\Collection to return rows without an associated entity. This objects implements the ArrayAccess interface to allow access the object as object->x or array[x].

### Methods

public *boolean* **offsetExists** (*int* \$index)

Checks whether an offset exists in the document

public *mixed* **offsetGet** (*string* \$index)

Returns the value of a field using the ArrayAccess interfase

public **offsetSet** (*string* \$index, *Phalcon\Mvc\ModelInterface* \$value)

Change a value using the ArrayAccess interface

public **offsetUnset** (*string* \$offset)

Rows cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

public *mixed* **readAttribute** (*string* \$attribute)

Reads an attribute value by its name

```
<?php

echo $robot->readAttribute('name');

public writeAttribute (string $attribute, mixed $value)
Writes an attribute value by its name

<?php

$robot->writeAttribute('name', 'Rosey');
```

## 2.54.148 Class Phalcon\Mvc\Collection\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Mvc\Collection\\* classes will use this class

### Methods

final private *Exception* \_\_clone () inherited from Exception

Clone the exception

public \_\_construct ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* getMessage () inherited from Exception

Gets the Exception message

final public *int* getCode () inherited from Exception

Gets the Exception code

final public *string* getFile () inherited from Exception

Gets the file in which the exception occurred

final public *int* getLine () inherited from Exception

Gets the line in which the exception occurred

final public *array* getTrace () inherited from Exception

Gets the stack trace

final public *Exception* getPrevious () inherited from Exception

Returns previous Exception

final public *Exception* getTraceAsString () inherited from Exception

Gets the stack trace as a string

public *string* \_\_toString () inherited from Exception

String representation of the exception

## 2.54.149 Class Phalcon\Mvc\Collection\Manager

*implements* [Phalcon\DI\InjectionAwareInterface](#), [Phalcon\Events\EventsAwareInterface](#),  
[Phalcon\Mvc\Collection\ManagerInterface](#)

This components controls the initialization of models, keeping record of relations between the different models of the application. A CollectionManager is injected to a model via a Dependency Injector Container such as Phalcon\DI.

<?php

```
$di = new Phalcon\DI();

$di->set('collectionManager', function(){
    return new Phalcon\Mvc\Collection\Manager();
});

$robot = new Robots($di);
```

### Methods

**public** **setDI** ([Phalcon\DiInterface](#) \$dependencyInjector)

Sets the DependencyInjector container

**public** [Phalcon\DiInterface](#) **getDI** ()

Returns the DependencyInjector container

**public** **setEventsManager** ([Phalcon\Events\ManagerInterface](#) \$eventsManager)

Sets the event manager

**public** [Phalcon\Events\ManagerInterface](#) **getEventsManager** ()

Returns the internal event manager

**public** **setCustomEventsManager** ([Phalcon\Mvc\CollectionInterface](#) \$model,  
[Phalcon\Events\ManagerInterface](#) \$eventsManager)

Sets a custom events manager for a specific model

**public** [Phalcon\Events\ManagerInterface](#) **getCustomEventsManager** ([Phalcon\Mvc\CollectionInterface](#) \$model)

Returns a custom events manager related to a model

**public** **initialize** ([Phalcon\Mvc\CollectionInterface](#) \$model)

Initializes a model in the models manager

**public** *bool* **isInitialized** (*string* \$modelName)

Check whether a model is already initialized

**public** [Phalcon\Mvc\CollectionInterface](#) **getLastInitialized** ()

Get the latest initialized model

**public** **setConnectionService** ([Phalcon\Mvc\CollectionInterface](#) \$model, *string* \$connectionService)

Sets a connection service for a specific model

**public** **useImplicitObjectIds** ([Phalcon\Mvc\CollectionInterface](#) \$model, *boolean* \$useImplicitObjectIds)

Sets if a model must use implicit objects ids

```
public boolean isUsingImplicitObjectIds (Phalcon\ Mvc\ CollectionInterface $model)
```

Checks if a model is using implicit object ids

```
public Phalcon\ Db\ AdapterInterface (?) MongoDB getConnection (Phalcon\ Mvc\ CollectionInterface $model)
```

Returns the connection related to a model

```
public notifyEvent (string $eventName, Phalcon\ Mvc\ CollectionInterface $model)
```

Receives events generated in the models and dispatches them to a events-manager if available Notify the behaviors that are listening in the model

## 2.54.150 Abstract class Phalcon\ Mvc\ Controller

*extends abstract class Phalcon\ DI\ Injectable*

*implements Phalcon\ Events\ EventsAwareInterface, Phalcon\ DI\ InjectionAwareInterface, Phalcon\ Mvc\ ControllerInterface*

Every application controller should extend this class that encapsulates all the controller functionality. The controllers provide the “flow” between models and views. Controllers are responsible for processing the incoming requests from the web browser, interrogating the models for data, and passing that data on to the views for presentation.

```
<?php
```

```
class PeopleController extends \Phalcon\ Mvc\ Controller
{
    //This action will be executed by default
    public function indexAction()
    {
    }

    public function findAction()
    {
    }

    public function saveAction()
    {
        //Forwards flow to the index action
        return $this-> dispatcher->forward(array('controller' => 'people', 'action' => 'index'));
    }
}
```

### Methods

final public **\_\_construct** ()

Phalcon\ Mvc\ Controller constructor

public **setDI** (*Phalcon\ DiInterface* \$dependencyInjector) inherited from Phalcon\ DI\ Injectable

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\DI\Injectable

Sets the event manager

public *Phalcon\Events\ManagerInterface* **getEventManager** () inherited from Phalcon\DI\Injectable

Returns the internal event manager

public **\_\_get** (*unknown* \$property) inherited from Phalcon\DI\Injectable

Magic method \_\_get

## 2.54.151 Class Phalcon\Mvc\Dispatcher

*extends* abstract class *Phalcon\Dispatcher*

*implements* *Phalcon\Events\EventsAwareInterface*, *Phalcon\DI\InjectionAwareInterface*, *Phalcon\DispatcherInterface*, *Phalcon\Mvc\DispatcherInterface*

Dispatching is the process of taking the request object, extracting the module name, controller name, action name, and optional parameters contained in it, and then instantiating a controller and calling an action of that controller.

<?php

```
$di = new Phalcon\DI();  
  
$dispatcher = new Phalcon\Mvc\Dispatcher();  
  
$dispatcher->setDI($di);  
  
$dispatcher->setControllerName('posts');  
$dispatcher->setActionName('index');  
$dispatcher->setParams(array());  
  
$controller = $dispatcher->dispatch();
```

### Constants

*integer* **EXCEPTION\_NO\_DI**

*integer* **EXCEPTION\_CYCLIC\_ROUTING**

*integer* **EXCEPTION\_HANDLER\_NOT\_FOUND**

*integer* **EXCEPTION\_INVALID\_HANDLER**

*integer* **EXCEPTION\_INVALID\_PARAMS**

*integer* **EXCEPTION\_ACTION\_NOT\_FOUND**

## Methods

public **setControllerSuffix** (*string* \$controllerSuffix)

Sets the default controller suffix

public **setDefaultController** (*string* \$controllerName)

Sets the default controller name

public **setControllerName** (*string* \$controllerName, [*unknown* \$isExact])

Sets the controller name to be dispatched

public *string* **getControllerName** ()

Gets last dispatched controller name

protected **\_throwDispatchException** ()

Throws an internal exception

protected **\_handleException** ()

Handles a user exception phalcon\_dispatcher\_fire\_event() first

public *string* **getControllerClass** ()

Possible controller class name that will be located to dispatch the request

public *Phalcon\ Mvc\ ControllerInterface* **getLastController** ()

Returns the lastest dispatched controller

public *Phalcon\ Mvc\ ControllerInterface* **getActiveController** ()

Returns the active controller in the dispatcher

public *string* **getPreviousControllerName** ()

Returns the previous controller in the dispatcher

public *string* **getPreviousActionName** ()

Returns the previous action in the dispatcher

public **\_\_construct** () inherited from Phalcon\Dispatcher

Phalcon\Dispatcher constructor

public **setDI** (*Phalcon\ DiInterface* \$dependencyInjector) inherited from Phalcon\Dispatcher

Sets the dependency injector

public *Phalcon\ DiInterface* **getDI** () inherited from Phalcon\Dispatcher

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\ Events\ ManagerInterface* \$eventsManager) inherited from Phalcon\Dispatcher

Sets the events manager

public *Phalcon\ Events\ ManagerInterface* **getEventsManager** () inherited from Phalcon\Dispatcher

Returns the internal event manager

public **setActionSuffix** (*string* \$actionSuffix) inherited from Phalcon\Dispatcher

Sets the default action suffix

public **setModuleName** (*string \$moduleName*) inherited from Phalcon\Dispatcher  
Sets the module where the controller is (only informative)

public *string getModuleName ()* inherited from Phalcon\Dispatcher  
Gets the module where the controller class is

public **setNamespaceName** (*string \$namespaceName*) inherited from Phalcon\Dispatcher  
Sets the namespace where the controller class is

public *string getNamespaceName ()* inherited from Phalcon\Dispatcher  
Gets a namespace to be prepended to the current handler name

public **setDefaultNamespace** (*string \$namespace*) inherited from Phalcon\Dispatcher  
Sets the default namespace

public *string getDefaultNamespace ()* inherited from Phalcon\Dispatcher  
Returns the default namespace

public **setDefaultAction** (*string \$actionName*) inherited from Phalcon\Dispatcher  
Sets the default action name

public **setActionName** (*string \$actionName*) inherited from Phalcon\Dispatcher  
Sets the action name to be dispatched

public *string getActionName ()* inherited from Phalcon\Dispatcher  
Gets the lastest dispatched action name

public **setParams** (*array \$params*) inherited from Phalcon\Dispatcher  
Sets action params to be dispatched

public *array getParams ()* inherited from Phalcon\Dispatcher  
Gets action params

public **setParam** (*mixed \$param, mixed \$value*) inherited from Phalcon\Dispatcher  
Set a param by its name or numeric index

public *mixedgetParam (mixed \$param, [string/array \$filters])* inherited from Phalcon\Dispatcher  
Gets a param by its name or numeric index

public *string getActiveMethod ()* inherited from Phalcon\Dispatcher  
Returns the current method to be/executed in the dispatcher

public *boolean isFinished ()* inherited from Phalcon\Dispatcher  
Checks if the dispatch loop is finished or has more pendent controllers/tasks to disptach

public **setReturnedValue** (*mixed \$value*) inherited from Phalcon\Dispatcher  
Sets the latest returned value by an action manually

public *mixed getReturnedValue ()* inherited from Phalcon\Dispatcher  
Returns value returned by the lastest dispatched action

public *object dispatch ()* inherited from Phalcon\Dispatcher  
Dispatches a handle action taking into account the routing parameters

public **forward** (*array* \$forward) inherited from Phalcon\Dispatcher

Forwards the execution flow to another controller/action Dispatchers are unique per module. Forwarding between modules is not allowed

<?php

```
$this->dispatcher->forward(array('controller' => 'posts', 'action' => 'index'));
```

public *boolean* **wasForwarded** () inherited from Phalcon\Dispatcher

Check if the current executed action was forwarded by another one

public *string* **getHandlerClass** () inherited from Phalcon\Dispatcher

Possible class name that will be located to dispatch the request

## 2.54.152 Class Phalcon\Mvc\Dispatcher\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Mvc\Dispatcher will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

## 2.54.153 Class Phalcon\Mvc\Micro

*extends abstract class Phalcon\DI\Injectable*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\DI\InjectionAwareInterface, ArrayAccess*

With Phalcon you can create “Micro-Framework like” applications. By doing this, you only need to write a minimal amount of code to create a PHP application. Micro applications are suitable to small applications, APIs and prototypes in a practical way.

```
<?php
```

```
$app = new Phalcon\Mvc\Micro();  
  
$app->get('/say/welcome/{name}', function ($name) {  
    echo "<h1>Welcome $name!</h1>";  
});  
  
$app->handle();
```

### Methods

public **\_\_construct** ([*Phalcon\DiInterface* \$dependencyInjector])

Phalcon\Mvc\Micro constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the DependencyInjector container

public *Phalcon\Mvc\Router\RouteInterface* **map** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler without any HTTP method constraint

public *Phalcon\Mvc\Router\RouteInterface* **get** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is GET

public *Phalcon\Mvc\Router\RouteInterface* **post** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is POST

public *Phalcon\Mvc\Router\RouteInterface* **put** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is PUT

public *Phalcon\Mvc\Router\RouteInterface* **patch** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is PATCH

public *Phalcon\Mvc\Router\RouteInterface* **head** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is HEAD

public *Phalcon\Mvc\Router\RouteInterface* **delete** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is DELETE

public *Phalcon\Mvc\Router\RouteInterface* **options** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is OPTIONS

public *Phalcon\Mvc\Micro* **mount** (*Phalcon\Mvc\Collection* \$collection)

Mounts a collection of handlers

public *Phalcon\ Mvc\ Micro* **notFound** (*callable* \$handler)

Sets a handler that will be called when the router doesn't match any of the defined routes

public *Phalcon\ Mvc\ RouterInterface* **getRouter** ()

Returns the internal router used by the application

public *Phalcon\ DI\ ServiceInterface* **setService** (*string* \$serviceName, *mixed* \$definition, [*boolean* \$shared])

Sets a service from the DI

public *boolean* **hasService** (*string* \$serviceName)

Checks if a service is registered in the DI

public *object* **getService** (*string* \$serviceName)

Obtains a service from the DI

public *mixed* **getSharedService** (*string* \$serviceName)

Obtains a shared service from the DI

public *mixed* **handle** ([*string* \$uri])

Handle the whole request

public **stop** ()

Stops the middleware execution avoiding than other middlewares be executed

public **setActiveHandler** (*callable* \$activeHandler)

Sets externally the handler that must be called by the matched route

public *callable* **getActiveHandler** ()

Return the handler that will be called for the matched route

public *mixed* **getReturnValue** ()

Returns the value returned by the executed handler

public *boolean* **offsetExists** (*unknown* \$serviceName)

Check if a service is registered in the internal services container using the array syntax. Alias for Phalcon\ Mvc\ Micro::hasService()

public **offsetSet** (*unknown* \$serviceName, *mixed* \$definition, [*unknown* \$shared])

Allows to register a shared service in the internal services container using the array syntax. Alias for Phalcon\ Mvc\ Micro::setService()

<?php

```
$app[‘request’] = new Phalcon\Http\Request();
```

public *mixed* **offsetGet** (*unknown* \$serviceName)

Allows to obtain a shared service in the internal services container using the array syntax. Alias for Phalcon\ Mvc\ Micro::getService()

<?php

```
var_dump($app[‘request’]);
```

public **offsetUnset** (*string \$alias*)

Removes a service from the internal services container using the array syntax

public *Phalcon\ Mvc\ Micro* **before** (*callable \$handler*)

Appends a before middleware to be called before execute the route

public *Phalcon\ Mvc\ Micro* **after** (*callable \$handler*)

Appends an ‘after’ middleware to be called after execute the route

public *Phalcon\ Mvc\ Micro* **finish** (*callable \$handler*)

Appends a ‘finish’ middleware to be called when the request is finished

public *array getHandlers* ()

Returns the internal handlers attached to the application

public *Phalcon\ DiInterface getDI* () inherited from *Phalcon\ DI\ Injectable*

Returns the internal dependency injector

public *setEventsManager* (*Phalcon\ Events\ ManagerInterface \$eventsManager*) inherited from *Phalcon\ DI\ Injectable*

Sets the event manager

public *Phalcon\ Events\ ManagerInterface getEventsManager* () inherited from *Phalcon\ DI\ Injectable*

Returns the internal event manager

public *\_\_get* (*unknown \$property*) inherited from *Phalcon\ DI\ Injectable*

Magic method *\_\_get*

## 2.54.154 Class *Phalcon\ Mvc\ Micro\ Collection*

*implements Phalcon\ Mvc\ Micro\ CollectionInterface*

Groups Micro-Mvc handlers as controllers

<?php

```
$app = new Phalcon\Mvc\Micro();

$collection = new Phalcon\Mvc\Micro\Collection();

$collection->setHandler(new PostsController());

$collection->get('/posts/edit/{id}', 'edit');

$app->mount($collection);
```

### Methods

public *Phalcon\ Mvc\ Micro\ CollectionInterface setPrefix* (*string \$prefix*)

Sets a prefix for all routes added to the collection

public *string getPrefix* ()

Returns the collection prefix if any

`public array getHandlers ()`

Returns the registered handlers

`public Phalcon\ Mvc\ Micro\ CollectionInterface setHandler (mixed $handler, [boolean $lazy])`

Sets the main handler

`public Phalcon\ Mvc\ Micro\ CollectionInterface setLazy (boolean $lazy)`

Sets if the main handler must be lazy loaded

`public boolean isLazy ()`

Returns if the main handler must be lazy loaded

`public mixed getHandler ()`

Returns the main handler

`public Phalcon\ Mvc\ Micro\ CollectionInterface map (string $routePattern, callable $handler)`

Maps a route to a handler

`public Phalcon\ Mvc\ Micro\ CollectionInterface get (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is GET

`public Phalcon\ Mvc\ Micro\ CollectionInterface post (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is POST

`public Phalcon\ Mvc\ Micro\ CollectionInterface put (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is PUT

`public Phalcon\ Mvc\ Micro\ CollectionInterface patch (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is PATCH

`public Phalcon\ Mvc\ Micro\ CollectionInterface head (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is HEAD

`public Phalcon\ Mvc\ Micro\ CollectionInterface delete (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is DELETE

`public Phalcon\ Mvc\ Micro\ CollectionInterface options (string $routePattern, callable $handler)`

Maps a route to a handler that only matches if the HTTP method is OPTIONS

## 2.54.155 Class Phalcon\ Mvc\ Micro\ Exception

`extends class Phalcon\ Exception`

Exceptions thrown in Phalcon\ Mvc\ Micro will use this class

### Methods

`final private Exception __clone ()` inherited from Exception

Clone the exception

`public __construct ([string $message], [int $code], [Exception $previous])` inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception  
Gets the Exception message

final public *int* **getCode** () inherited from Exception  
Gets the Exception code

final public *string* **getFile** () inherited from Exception  
Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception  
Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception  
Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception  
Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception  
Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception  
String representation of the exception

## 2.54.156 Class Phalcon\Mvc\Micro\LazyLoader

Lazy-Load of handlers for Mvc\Micro using auto-loading

### Methods

public **\_\_construct** (*string* \$definition)  
Phalcon\Mvc\Micro\LazyLoader constructor

public *mixed* **\_\_call** (*string* \$method, *array* \$arguments)  
Initializes the internal handler, calling functions on it

## 2.54.157 Abstract class Phalcon\Mvc\Model

*implements* [Phalcon\ Mvc\ ModelInterface](#), [Phalcon\ Mvc\ Model\ ResultInterface](#),  
[Phalcon\ DI\ InjectionAwareInterface](#), [Serializable](#)

Phalcon\Mvc\Model connects business objects and database tables to create a persistable domain model where logic and data are presented in one wrapping. It's an implementation of the object-relational mapping (ORM). A model represents the information (data) of the application and the rules to manipulate that data. Models are primarily used for managing the rules of interaction with a corresponding database table. In most cases, each table in your database will correspond to one model in your application. The bulk of your application's business logic will be concentrated in the models. Phalcon\Mvc\Model is the first ORM written in C-language for PHP, giving to developers high performance when interacting with databases while is also easy to use.

```
<?php

$robot = new Robots();
$robot->type = 'mechanical';
$robot->name = 'Astro Boy';
$robot->year = 1952;
if ($robot->save() == false) {
    echo "Umh, We can store robots: ";
    foreach ($robot->getMessages() as $message) {
        echo $message;
    }
} else {
    echo "Great, a new robot was saved successfully!";
}
```

## Constants

*integer* **OP\_NONE**  
*integer* **OP\_CREATE**  
*integer* **OP\_UPDATE**  
*integer* **OP\_DELETE**  
*integer* **DIRTY\_STATE\_PERSISTENT**  
*integer* **DIRTY\_STATE\_TRANSIENT**  
*integer* **DIRTY\_STATE\_DETACHED**

## Methods

final public **\_\_construct** ([*Phalcon\DiInterface* \$dependencyInjector], [*Phalcon\Mvc\Model\ManagerInterface* \$modelsManager])  
Phalcon\ Mvc\ Model constructor

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)  
Sets the dependency injection container

public *Phalcon\DiInterface* **getDI** ()  
Returns the dependency injection container

protected **setEventsManager** ()  
Sets a custom events manager

protected *Phalcon\Events\ManagerInterface* **getEventsManager** ()  
Returns the custom events manager

public *Phalcon\Mvc\Model\MetaDataInterface* **getModelsMetaData** ()  
Returns the models meta-data service related to the entity instance

public *Phalcon\Mvc\Model\ManagerInterface* **getModelsManager** ()  
Returns the models manager related to the entity instance

public *Phalcon\Mvc\Model* **setTransaction** (*Phalcon\Mvc\Model\TransactionInterface* \$transaction)

Sets a transaction related to the Model instance

```
<?php

use Phalcon\Mvc\Model\Transaction\Manager as TxManager;
use Phalcon\Mvc\Model\Transaction\Failed as TxFailed;

try {

    $txManager = new TxManager();

    $transaction = $txManager->get();

    $robot = new Robots();
    $robot->setTransaction($transaction);
    $robot->name = 'WALL·E';
    $robot->created_at = date('Y-m-d');
    if ($robot->save() == false) {
        $transaction->rollback("Can't save robot");
    }

    $robotPart = new RobotParts();
    $robotPart->setTransaction($transaction);
    $robotPart->type = 'head';
    if ($robotPart->save() == false) {
        $transaction->rollback("Robot part cannot be saved");
    }

    $transaction->commit();

} catch (TxFailed $e) {
    echo 'Failed, reason: ', $e->getMessage();
}
```

protected *Phalcon|Mvc|Model* **setSource** (*string* \$source)

Sets table name which model should be mapped

public *string* **getSource** ()

Returns table name mapped in the model

protected *Phalcon|Mvc|Model* **setSchema** (*string* \$schema)

Sets schema name where table mapped is located

public *string* **getSchema** ()

Returns schema name where table mapped is located

public *Phalcon|Mvc|Model* **setConnectionService** (*string* \$connectionService)

Sets the DependencyInjection connection service name

public *Phalcon|Mvc|Model* **setReadConnectionService** (*string* \$connectionService)

Sets the DependencyInjection connection service name used to read data

public *Phalcon|Mvc|Model* **setWriteConnectionService** (*string* \$connectionService)

Sets the DependencyInjection connection service name used to write data

public *string* **getReadConnectionService** ()

Returns the DependencyInjection connection service name used to read data related the model

```
public string getWriteConnectionService ()
```

Returns the DependencyInjection connection service name used to write data related to the model

```
public Phalcon\ Mvc\ Model setDirtyState (int $dirtyState)
```

Sets the dirty state of the object using one of the DIRTY\_STATE\_\* constants

```
public int getDirtyState ()
```

Returns one of the DIRTY\_STATE\_\* constants telling if the record exists in the database or not

```
public Phalcon\ Db\ AdapterInterface getReadConnection ()
```

Gets the connection used to read data for the model

```
public Phalcon\ Db\ AdapterInterface getWriteConnection ()
```

Gets the connection used to write data to the model

```
public Phalcon\ Mvc\ Model assign (array $data, [array $columnMap])
```

Assigns values to a model from an array

```
<?php
```

```
$robot->assign(array(
    'type' => 'mechanical',
    'name' => 'Astro Boy',
    'year' => 1952
));
```

```
public static Phalcon\ Mvc\ Model cloneResultMap (Phalcon\ Mvc\ Model $base, array $data, array $columnMap, [int $dirtyState], [boolean $keepSnapshots])
```

Assigns values to a model from an array returning a new model.

```
<?php
```

```
$robot = \Phalcon\ Mvc\ Model::cloneResultMap(new Robots(), array(
    'type' => 'mechanical',
    'name' => 'Astro Boy',
    'year' => 1952
));
```

```
public static mixed cloneResultMapHydrate (array $data, array $columnMap, int $hydrationMode)
```

Returns an hydrated result based on the data and the column map

```
public static Phalcon\ Mvc\ Model cloneResult (Phalcon\ Mvc\ Model $base, array $data, [int $dirtyState])
```

Assigns values to a model from an array returning a new model

```
<?php
```

```
$robot = Phalcon\ Mvc\ Model::cloneResult(new Robots(), array(
    'type' => 'mechanical',
    'name' => 'Astro Boy',
    'year' => 1952
));
```

```
public static Phalcon\ Mvc\ Model\ ResultsetInterface find ([array $parameters])
```

Allows to query a set of records that match the specified conditions

```
<?php
```

```
//How many robots are there?  
$robots = Robots::find();  
echo "There are ", count($robots), "\n";  
  
//How many mechanical robots are there?  
$robots = Robots::find("type='mechanical'");  
echo "There are ", count($robots), "\n";  
  
//Get and print virtual robots ordered by name  
$robots = Robots::find(array("type='virtual'", "order" => "name"));  
foreach ($robots as $robot) {  
    echo $robot->name, "\n";  
}  
  
//Get first 100 virtual robots ordered by name  
$robots = Robots::find(array("type='virtual'", "order" => "name", "limit" => 100));  
foreach ($robots as $robot) {  
    echo $robot->name, "\n";  
}
```

public static *Phalcon\ Mvc\ Model* **findFirst** ([array \$parameters])

Allows to query the first record that match the specified conditions

```
<?php
```

```
//What's the first robot in robots table?  
$robot = Robots::findFirst();  
echo "The robot name is ", $robot->name;  
  
//What's the first mechanical robot in robots table?  
$robot = Robots::findFirst("type='mechanical'");  
echo "The first mechanical robot name is ", $robot->name;  
  
//Get first virtual robot ordered by name  
$robot = Robots::findFirst(array("type='virtual'", "order" => "name"));  
echo "The first virtual robot name is ", $robot->name;
```

public static *Phalcon\ Mvc\ Model\ Criteria* **query** ([*Phalcon\ DiInterface* \$dependencyInjector])

Create a criteria for a specific model

protected *boolean* **\_exists** ()

Checks if the current record already exists or not

protected static *Phalcon\ Mvc\ Model\ ResultsetInterface* **\_groupResult** ()

Generate a PHQL SELECT statement for an aggregate

public static *int* **count** ([array \$parameters])

Allows to count how many records match the specified conditions

```
<?php
```

```
//How many robots are there?  
$number = Robots::count();  
echo "There are ", $number, "\n";
```

```
//How many mechanical robots are there?
$number = Robots::count("type='mechanical'");
echo "There are ", $number, " mechanical robots\n";
```

**public static double sum ([array \$parameters])**

Allows to calculate a summatory on a column that match the specified conditions

<?php

```
//How much are all robots?
$sum = Robots::sum(array('column' => 'price'));
echo "The total price of robots is ", $sum, "\n";

//How much are mechanical robots?
$sum = Robots::sum(array("type='mechanical'", 'column' => 'price'));
echo "The total price of mechanical robots is ", $sum, "\n";
```

**public static mixed maximum ([array \$parameters])**

Allows to get the maximum value of a column that match the specified conditions

<?php

```
//What is the maximum robot id?
$id = Robots::maximum(array('column' => 'id'));
echo "The maximum robot id is: ", $id, "\n";

//What is the maximum id of mechanical robots?
$sum = Robots::maximum(array("type='mechanical'", 'column' => 'id'));
echo "The maximum robot id of mechanical robots is ", $id, "\n";
```

**public static mixed minimum ([array \$parameters])**

Allows to get the minimum value of a column that match the specified conditions

<?php

```
//What is the minimum robot id?
$id = Robots::minimum(array('column' => 'id'));
echo "The minimum robot id is: ", $id;

//What is the minimum id of mechanical robots?
$sum = Robots::minimum(array("type='mechanical'", 'column' => 'id'));
echo "The minimum robot id of mechanical robots is ", $id;
```

**public static double average ([array \$parameters])**

Allows to calculate the average value on a column matching the specified conditions

<?php

```
//What's the average price of robots?
$average = Robots::average(array('column' => 'price'));
echo "The average price is ", $average, "\n";

//What's the average price of mechanical robots?
$average = Robots::average(array("type='mechanical'", 'column' => 'price'));
echo "The average price of mechanical robots is ", $average, "\n";
```

**public boolean fireEvent (string \$eventName)**

Fires an event, implicitly calls behaviors and listeners in the events manager are notified

```
public boolean fireEventCancel (string $eventName)
```

Fires an event, implicitly calls behaviors and listeners in the events manager are notified This method stops if one of the callbacks/listeners returns boolean false

```
protected boolean _cancelOperation ()
```

Cancel the current operation

```
public Phalcon\ Mvc\ Model appendMessage (Phalcon\ Mvc\ Model\ MessageInterface $message)
```

Appends a customized message on the validation process

```
<?php
```

```
use \Phalcon\ Mvc\ Model\ Message as Message;

class Robots extends Phalcon\ Mvc\ Model
{

    public function beforeSave()
    {
        if ($this->name == 'Peter') {
            $message = new Message("Sorry, but a robot cannot be named Peter");
            $this->appendMessage($message);
        }
    }
}
```

```
protected Phalcon\ Mvc\ Model validate (Phalcon\ Mvc\ Model\ ValidatorInterface $validator)
```

Executes validators on every validation call

```
<?php
```

```
use Phalcon\ Mvc\ Model\ Validator\ ExclusionIn as ExclusionIn;

class Subscriptors extends Phalcon\ Mvc\ Model
{

    public function validation()
    {
        $this->validate(new ExclusionIn(array(
            'field' => 'status',
            'domain' => array('A', 'I')
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

```
public boolean validationHasFailed ()
```

Check whether validation process has generated any messages

```
<?php
```

```
use Phalcon\ Mvc\ Model\ Validator\ ExclusionIn as ExclusionIn;
```

```

class Subscribers extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new ExclusionIn(array(
            'field' => 'status',
            'domain' => array('A', 'I')
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}

```

`public Phalcon\Mvc\Model\MessageInterface [] getMessages ([unknown $filter])`

Returns all the validation messages

<?php

```

$robot = new Robots();
$robot->type = 'mechanical';
$robot->name = 'Astro Boy';
$robot->year = 1952;
if ($robot->save() == false) {
    echo "Umh, We can't store robots right now ";
    foreach ($robot->getMessages() as $message) {
        echo $message;
    }
} else {
    echo "Great, a new robot was saved successfully!";
}

```

`protected boolean _checkForeignKeysRestrict ()`

Reads “belongs to” relations and check the virtual foreign keys when inserting or updating records to verify that inserted/updated values are present in the related entity

`protected boolean _checkForeignKeysReverseRestrict ()`

Reads both “hasMany” and “hasOne” relations and checks the virtual foreign keys (restrict) when deleting records

`protected boolean _checkForeignKeysReverseCascade ()`

Reads both “hasMany” and “hasOne” relations and checks the virtual foreign keys (cascade) when deleting records

`protected boolean _preSave ()`

Executes internal hooks before save a record

`protected boolean _postSave ()`

Executes internal events after save a record

`protected boolean _doLowInsert ()`

Sends a pre-build INSERT SQL statement to the relational database system

`protected boolean _doLowUpdate ()`

Sends a pre-build UPDATE SQL statement to the relational database system

```
protected boolean _preSaveRelatedRecords ()
```

Saves related records that must be stored prior to save the master record

```
protected boolean _postSaveRelatedRecords ()
```

Save the related records assigned in the has-one/has-many relations

```
public boolean save ([array $data], [array $whiteList])
```

Inserts or updates a model instance. Returning true on success or false otherwise.

```
<?php
```

```
//Creating a new robot
```

```
$robot = new Robots();  
$robot->type = 'mechanical';  
$robot->name = 'Astro Boy';  
$robot->year = 1952;  
$robot->save();
```

```
//Updating a robot name
```

```
$robot = Robots::findFirst("id=100");  
$robot->name = "Biomass";  
$robot->save();
```

```
public boolean create ([array $data], [array $whiteList])
```

Inserts a model instance. If the instance already exists in the persistance it will throw an exception Returning true on success or false otherwise.

```
<?php
```

```
//Creating a new robot
```

```
$robot = new Robots();  
$robot->type = 'mechanical';  
$robot->name = 'Astro Boy';  
$robot->year = 1952;  
$robot->create();
```

```
//Passing an array to create
```

```
$robot = new Robots();  
$robot->create(array(  
    'type' => 'mechanical',  
    'name' => 'Astroy Boy',  
    'year' => 1952  
));
```

```
public boolean update ([array $data], [array $whiteList])
```

Updates a model instance. If the instance doesn't exist in the persistance it will throw an exception Returning true on success or false otherwise.

```
<?php
```

```
//Updating a robot name
```

```
$robot = Robots::findFirst("id=100");  
$robot->name = "Biomass";  
$robot->update();
```

**public boolean delete ()**

Deletes a model instance. Returning true on success or false otherwise.

<?php

```
$robot = Robots::findFirst("id=100");
$robot->delete();

foreach (Robots::find("type = 'mechanical'") as $robot) {
    $robot->delete();
}
```

**public int getOperationMade ()**

Returns the type of the latest operation performed by the ORM Returns one of the OP\_\* class constants

**public refresh ()**

Refreshes the model attributes re-querying the record from the database

**public skipOperation (boolean \$skip)**

Skips the current operation forcing a success state

**public mixed readAttribute (string \$attribute)**

Reads an attribute value by its name

<?php

```
echo $robot->readAttribute('name');
```

**public writeAttribute (string \$attribute, mixed \$value)**

Writes an attribute value by its name

<?php

```
$robot->writeAttribute('name', 'Rosey');
```

**protected skipAttributes (array \$attributes, [boolean \$replace])**

Sets a list of attributes that must be skipped from the generated INSERT/UPDATE statement

<?php

```
class Robots extends \Phalcon\Mvc\Model
{
```

```
    public function initialize()
    {
        $this->skipAttributes(array('price'));
    }
}
```

**protected skipAttributesOnCreate (array \$attributes, [boolean \$replace])**

Sets a list of attributes that must be skipped from the generated INSERT statement

<?php

```
class Robots extends \Phalcon\Mvc\Model
```

```
{  
  
    public function initialize()  
    {  
        $this->skipAttributesOnCreate(array('created_at'));  
    }  
  
}  
  
protected skipAttributesOnUpdate (array $attributes, [boolean $replace])  
Sets a list of attributes that must be skipped from the generated UPDATE statement  
<?php  
  
class Robots extends \Phalcon\Mvc\Model  
{  
  
    public function initialize()  
    {  
        $this->skipAttributesOnUpdate(array('modified_in'));  
    }  
  
}  
  
public Phalcon\Mvc\Model\Relation hasOne (mixed $fields, string $referenceModel, mixed  
$referencedFields, [array $options])  
Setup a 1-1 relation between two models  
<?php  
  
class Robots extends \Phalcon\Mvc\Model  
{  
  
    public function initialize()  
    {  
        $this->hasOne('id', 'RobotsDescription', 'robots_id');  
    }  
  
}  
  
public Phalcon\Mvc\Model\Relation belongsTo (mixed $fields, string $referenceModel, mixed  
$referencedFields, [array $options])  
Setup a relation reverse 1-1 between two models  
<?php  
  
class RobotsParts extends \Phalcon\Mvc\Model  
{  
  
    public function initialize()  
    {  
        $this->belongsTo('robots_id', 'Robots', 'id');  
    }  
  
}  
  
public Phalcon\Mvc\Model\Relation hasMany (mixed $fields, string $referenceModel, mixed  
$referencedFields, [array $options])
```

Setup a relation 1-n between two models

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->hasMany('id', 'RobotsParts', 'robots_id');
    }

}

public Phalcon\Mvc\Model\Relation hasManyToMany (string $fields, string $intermediateModel, string $intermediateFields, string $intermediateReferencedFields, unknown $referenceModel, string $referencedFields, [array $options])
```

Setup a relation n-n between two models through an intermediate relation

```
<?php

class Robots extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        //Setup a many-to-many relation to Parts through RobotsParts
        $this->hasManyToMany(
            'id',
            'RobotsParts',
            'robots_id',
            'parts_id',
            'Parts',
            'id'
        );
    }

}
```

public addBehavior (Phalcon\Mvc\Model\BehaviorInterface \$behavior)

Setups a behavior in a model

```
<?php

use Phalcon\Mvc\Model\Behavior\Timestampable;

class Robots extends \Phalcon\Mvc\Model
{

    public function initialize()
    {
        $this->addBehavior(new Timestampable(array(
            'onCreate' => array(
                'field' => 'created_at',
                'format' => 'Y-m-d'
            )
        )));
    }

}
```

}

protected **keepSnapshots** (*boolean* \$keepSnapshots)

Sets if the model must keep the original record snapshot in memory

<?php

```
class Robots extends \Phalcon\Mvc\Model
{
```

```
    public function initialize()
    {
        $this->keepSnapshots(true);
    }
```

}

public **setSnapshotData** (*array* \$data, [*array* \$columnMap])

Sets the record's snapshot data. This method is used internally to set snapshot data when the model was set up to keep snapshot data

public *boolean* **hasSnapshotData** ()

Checks if the object has internal snapshot data

public *array* **getSnapshotData** ()

Returns the internal snapshot data

public **hasChanged** ([*boolean* \$fieldName])

Check if a specific attribute has changed This only works if the model is keeping data snapshots

public *array* **getChangedFields** ()

Returns a list of changed values

protected **useDynamicUpdate** (*boolean* \$dynamicUpdate)

Sets if a model must use dynamic update instead of the all-field update

<?php

```
class Robots extends \Phalcon\Mvc\Model
{
```

```
    public function initialize()
    {
        $this->useDynamicUpdate(true);
    }
```

}

public *Phalcon\ Mvc\ Model\ ResultsetInterface* **getRelated** (*string* \$alias, [*array* \$arguments])

Returns related records based on defined relations

protected *mixed* **\_getRelatedRecords** ()

Returns related records defined relations depending on the method name

public *mixed* **\_\_call** (*string* \$method, [*array* \$arguments])

Handles method calls when a method is not implemented

```
public static mixed __callStatic (string $method, [array $arguments])
```

Handles method calls when a static method is not implemented

```
public __set (string $property, mixed $value)
```

Magic method to assign values to the the model

```
public Phalcon\|Mvc\|Model\|Resultset __get (string $property)
```

Magic method to get related records using the relation alias as a property

```
public __isSet (string $property)
```

Magic method to check if a property is a valid relation

```
public string serialize ()
```

Serializes the object ignoring connections, services, related objects or static properties

```
public unserialize (string $data)
```

Unserializes the object from a serialized string

```
public array dump ()
```

Returns a simple representation of the object that can be used with var\_dump

```
<?php
```

```
var_dump($robot->dump());
```

```
public array toArray ([array $columns])
```

Returns the instance as an array representation

```
<?php
```

```
print_r($robot->toArray());
```

```
public static setup (array $options)
```

Enables/disables options in the ORM Available options: events — Enables/Disables globally the internal events virtualForeignKeys — Enables/Disables virtual foreign keys columnRenaming — Enables/Disables column renaming notNullValidations — Enables/Disables automatic not null validation exceptionOnFailedSave — Enables/Disables throws an exception if the saving process fails phqlLiterals — Enables/Disables literals in PHQL this improves the security of applications

## 2.54.158 Abstract class Phalcon\|Mvc\|Model\|Behavior

*implements Phalcon\|Mvc\|Model\|BehaviorInterface*

This is an optional base class for ORM behaviors

### Methods

```
public __construct ([array $options])
```

```
protected mustTakeAction ()
```

Checks whether the behavior must take action on certain event

protected *array* **getOptions** ()

Returns the behavior options related to an event

public **notify** (*string* \$type, *Phalcon\ Mvc\ ModelInterface* \$model)

This method receives the notifications from the EventsManager

public **missingMethod** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$method, [*array* \$arguments])

Acts as fallbacks when a missing method is called on the model

## 2.54.159 Class Phalcon\ Mvc\ Model\ Behavior\ SoftDelete

*extends* abstract class *Phalcon\ Mvc\ Model\ Behavior*

*implements* *Phalcon\ Mvc\ Model\ BehaviorInterface*

Instead of permanently delete a record it marks the record as deleted changing the value of a flag column

### Methods

public **notify** (*string* \$type, *Phalcon\ Mvc\ ModelInterface* \$model)

Listens for notifications from the models manager

public **\_\_construct** ([*array* \$options]) inherited from *Phalcon\ Mvc\ Model\ Behavior*

*Phalcon\ Mvc\ Model\ Behavior*

protected **mustTakeAction** () inherited from *Phalcon\ Mvc\ Model\ Behavior*

Checks whether the behavior must take action on certain event

protected *array* **getOptions** () inherited from *Phalcon\ Mvc\ Model\ Behavior*

Returns the behavior options related to an event

public **missingMethod** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$method, [*array* \$arguments])  
inherited from *Phalcon\ Mvc\ Model\ Behavior*

Acts as fallbacks when a missing method is called on the model

## 2.54.160 Class Phalcon\ Mvc\ Model\ Behavior\ Timestampable

*extends* abstract class *Phalcon\ Mvc\ Model\ Behavior*

*implements* *Phalcon\ Mvc\ Model\ BehaviorInterface*

Allows to automatically update a model's attribute saving the datetime when a record is created or updated

### Methods

public **notify** (*string* \$type, *Phalcon\ Mvc\ ModelInterface* \$model)

Listens for notifications from the models manager

public **\_\_construct** ([*array* \$options]) inherited from *Phalcon\ Mvc\ Model\ Behavior*

*Phalcon\ Mvc\ Model\ Behavior*

protected **mustTakeAction** () inherited from *Phalcon\ Mvc\ Model\ Behavior*

Checks whether the behavior must take action on certain event

protected **array** **getOptions** () inherited from Phalcon\Mvc\Model\Behavior

Returns the behavior options related to an event

public **missingMethod** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$method, [*array* \$arguments])  
inherited from Phalcon\Mvc\Model\Behavior

Acts as fallbacks when a missing method is called on the model

## 2.54.161 Class Phalcon\Mvc\Model\Criteria

*implements Phalcon\ Mvc\ Model\ CriteriaInterface, Phalcon\ DI\ InjectionAwareInterface*

This class allows to build the array parameter required by Phalcon\Mvc\Model::find and Phalcon\Mvc\Model::findFirst using an object-oriented interface

<?php

```
$robots = Robots::query()
->where("type = :type:")
->andWhere("year < 2000")
->bind(array("type" => "mechanical"))
->order("name")
->execute();
```

### Methods

public **setDI** (*Phalcon\ DiInterface* \$dependencyInjector)

Sets the DependencyInjector container

public *Phalcon\ DiInterface* **getDI** ()

Returns the DependencyInjector container

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **set modelName** (*string* \$modelName)

Set a model on which the query will be executed

public *string* **getModelName** ()

Returns an internal model name on which the criteria will be applied

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **bind** (*string* \$bindParams)

Sets the bound parameters in the criteria This method replaces all previously set bound parameters

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **bindTypes** (*string* \$bindTypes)

Sets the bind types in the criteria This method replaces all previously set bound parameters

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **columns** (*string/array* \$columns)

Sets the columns to be queried

<?php

```
$criteria->columns(array('id', 'name'));
```

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **join** (*string \$model*, [*string \$conditions*], [*string \$alias*], [*string \$type*])

Adds a join to the query

```
<?php
```

```
$criteria->join('Robots');  
$criteria->join('Robots', 'r.id = RobotsParts.robots_id');  
$criteria->join('Robots', 'r.id = RobotsParts.robots_id', 'r');  
$criteria->join('Robots', 'r.id = RobotsParts.robots_id', 'r', 'LEFT');
```

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **innerJoin** (*string \$model*, [*string \$conditions*], [*string \$alias*])

Adds a INNER join to the query

```
<?php
```

```
$criteria->innerJoin('Robots');  
$criteria->innerJoin('Robots', 'r.id = RobotsParts.robots_id');  
$criteria->innerJoin('Robots', 'r.id = RobotsParts.robots_id', 'r');  
$criteria->innerJoin('Robots', 'r.id = RobotsParts.robots_id', 'r', 'LEFT');
```

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **leftJoin** (*string \$model*, [*string \$conditions*], [*string \$alias*])

Adds a LEFT join to the query

```
<?php
```

```
$criteria->leftJoin('Robots', 'r.id = RobotsParts.robots_id', 'r');
```

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **rightJoin** (*string \$model*, [*string \$conditions*], [*string \$alias*])

Adds a RIGHT join to the query

```
<?php
```

```
$criteria->rightJoin('Robots', 'r.id = RobotsParts.robots_id', 'r');
```

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **where** (*string \$conditions*)

Sets the conditions parameter in the criteria

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **addWhere** (*string \$conditions*, [*array \$bindParams*], [*array \$bindTypes*])

Appends a condition to the current conditions using an AND operator (deprecated)

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **andWhere** (*string \$conditions*, [*array \$bindParams*], [*array \$bindTypes*])

Appends a condition to the current conditions using an AND operator

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **orWhere** (*string \$conditions*, [*array \$bindParams*], [*array \$bindTypes*])

Appends a condition to the current conditions using an OR operator

public *Phalcon\ Mvc\ Model\ CriteriaInterface* **betweenWhere** (*string \$expr*, *mixed \$minimum*, *mixed \$maximum*)

Appends a BETWEEN condition to the current conditions

```
<?php

$criteria->betweenWhere('price', 100.25, 200.50);

public Phalcon\ Mvc\ Model\ CriteriaInterface notBetweenWhere (string $expr, mixed $minimum, mixed
$maximum)

Appends a NOT BETWEEN condition to the current conditions

<?php

$criteria->notBetweenWhere('price', 100.25, 200.50);

public Phalcon\ Mvc\ Model\ CriteriaInterface inWhere (string $expr, array $values)

Appends an IN condition to the current conditions

<?php

$criteria->inWhere('id', [1, 2, 3]);

public Phalcon\ Mvc\ Model\ CriteriaInterface notInWhere (string $expr, array $values)

Appends a NOT IN condition to the current conditions

<?php

$criteria->notInWhere('id', [1, 2, 3]);

public Phalcon\ Mvc\ Model\ CriteriaInterface conditions (string $conditions)

Adds the conditions parameter to the criteria

public Phalcon\ Mvc\ Model\ CriteriaInterface order (string $orderColumns)

Adds the order-by parameter to the criteria (deprecated)

public Phalcon\ Mvc\ Model\ CriteriaInterface orderBy (string $orderColumns)

Adds the order-by parameter to the criteria

public Phalcon\ Mvc\ Model\ CriteriaInterface limit (int $limit, [int $offset])

Adds the limit parameter to the criteria

public Phalcon\ Mvc\ Model\ CriteriaInterface forUpdate ([boolean $forUpdate])

Adds the “for_update” parameter to the criteria

public Phalcon\ Mvc\ Model\ CriteriaInterface sharedLock ([boolean $sharedLock])

Adds the “shared_lock” parameter to the criteria

public string getWhere ()

Returns the conditions parameter in the criteria

public string/array getColumns ()

Return the columns to be queried

public string getConditions ()

Returns the conditions parameter in the criteria

public string getLimit ()
```

Returns the limit parameter in the criteria

`public string getOrder ()`

Returns the order parameter in the criteria

`public array getParams ()`

Returns all the parameters defined in the criteria

`public static Phalcon\ Mvc\ Model\ Criteria fromInput (Phalcon\ DiInterface $dependencyInjector, string $modelName, array $data)`

Builds a Phalcon\ Mvc\ Model\ Criteria based on an input array like \$\_POST

`public Phalcon\ Mvc\ Model\ ResultsetInterface execute ()`

Executes a find using the parameters built with the criteria

`public Phalcon\ Mvc\ Model\ CriteriaInterface cache (unknown $option)`

Sets the cache options in the criteria This method replaces all previously set cache options

## 2.54.162 Class Phalcon\ Mvc\ Model\ Exception

`extends class Phalcon\ Exception`

Exceptions thrown in Phalcon\ Mvc\ Model\ \* classes will use this class

### Methods

`final private Exception __clone ()` inherited from Exception

Clone the exception

`public __construct ([string $message], [int $code], [Exception $previous])` inherited from Exception

Exception constructor

`final public string getMessage ()` inherited from Exception

Gets the Exception message

`final public int getCode ()` inherited from Exception

Gets the Exception code

`final public string getFile ()` inherited from Exception

Gets the file in which the exception occurred

`final public int getLine ()` inherited from Exception

Gets the line in which the exception occurred

`final public array getTrace ()` inherited from Exception

Gets the stack trace

`final public Exception getPrevious ()` inherited from Exception

Returns previous Exception

`final public Exception getTraceAsString ()` inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString()** inherited from Exception  
 String representation of the exception

### 2.54.163 Class Phalcon\Mvc\Model\Manager

*implements* *Phalcon\Mvc\Model\ManagerInterface*, *Phalcon\DI\InjectionAwareInterface*, *Phalcon\Events\EventsAwareInterface*

This components controls the initialization of models, keeping record of relations between the different models of the application. A ModelsManager is injected to a model via a Dependency Injector/Services Container such as Phalcon\DI.

```
<?php

$di = new Phalcon\DI();

$di->set('modelsManager', function() {
    return new Phalcon\Mvc\Model\Manager();
});

$robot = new Robots($di);
```

#### Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the DependencyInjector container

public *Phalcon\DiInterface* **getDI** ()

Returns the DependencyInjector container

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager)

Sets a global events manager

public *Phalcon\Events\ManagerInterface* **getEventsManager** ()

Returns the internal event manager

public **setCustomEventsManager** (*Phalcon\Mvc\ModelInterface* \$model, *Phalcon\Events\ManagerInterface* \$eventsManager)

Sets a custom events manager for a specific model

public *Phalcon\Events\ManagerInterface* **getCustomEventsManager** (*Phalcon\Mvc\ModelInterface* \$model)

Returns a custom events manager related to a model

public *boolean* **initialize** (*Phalcon\Mvc\ModelInterface* \$model)

Initializes a model in the model manager

public *bool* **isInitialized** (*string* \$modelName)

Check whether a model is already initialized

public *Phalcon\Mvc\ModelInterface* **getLastInitialized** ()

Get last initialized model

public *Phalcon\ Mvc\ ModelInterface* **load** (*string* \$modelName, *boolean* \$newInstance)

Loads a model throwing an exception if it doesn't exist

public *string* **setModelSource** (*Phalcon\ Mvc\ Model* \$model, *string* \$source)

Sets the mapped source for a model

public *string* **getModelSource** (*Phalcon\ Mvc\ Model* \$model)

Returns the mapped source for a model

public *string* **setModelSchema** (*Phalcon\ Mvc\ Model* \$model, *string* \$schema)

Sets the mapped schema for a model

public *string* **getModelSchema** (*Phalcon\ Mvc\ Model* \$model)

Returns the mapped schema for a model

public **setConnectionService** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$connectionService)

Sets both write and read connection service for a model

public **setWriteConnectionService** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$connectionService)

Sets write connection service for a model

public **setReadConnectionService** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$connectionService)

Sets read connection service for a model

public *Phalcon\ Db\ AdapterInterface* **getWriteConnection** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the connection to write data related to a model

public *Phalcon\ Db\ AdapterInterface* **getReadConnection** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the connection to read data related to a model

public **getReadConnectionService** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the connection service name used to read data related to a model

public **getWriteConnectionService** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the connection service name used to write data related to a model

public **notifyEvent** (*string* \$eventName, *Phalcon\ Mvc\ ModelInterface* \$model)

Receives events generated in the models and dispatches them to a events-manager if available Notify the behaviors that are listening in the model

public *boolean* **missingMethod** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$eventName, *array* \$data)

Dispatch a event to the listeners and behaviors This method expects that the endpoint listeners/behaviors returns true meaning that a least one is implemented

public **addBehavior** (*Phalcon\ Mvc\ ModelInterface* \$model, *Phalcon\ Mvc\ Model\ BehaviorInterface* \$behavior)

Binds a behavior to a model

public **keepSnapshots** (*Phalcon\ Mvc\ Model* \$model, *boolean* \$keepSnapshots)

Sets if a model must keep snapshots

public *boolean* **isKeepingSnapshots** (*unknown* \$model)

Checks if a model is keeping snapshots for the queried records

**public `useDynamicUpdate`** (*Phalcon\ Mvc\ Model* \$model, *boolean* \$dynamicUpdate)

Sets if a model must use dynamic update instead of the all-field update

**public `isUsingDynamicUpdate`** (*unknown* \$model)

Checks if a model is using dynamic update instead of all-field update

**public `Phalcon\ Mvc\ Model\ Relation addHasOne`** (*Phalcon\ Mvc\ Model* \$model, *mixed* \$fields, *string* \$referencedModel, *mixed* \$referencedFields, [*array* \$options])

Setup a 1-1 relation between two models

**public `Phalcon\ Mvc\ Model\ Relation addBelongsTo`** (*Phalcon\ Mvc\ Model* \$model, *mixed* \$fields, *string* \$referencedModel, *mixed* \$referencedFields, [*array* \$options])

Setup a relation reverse many to one between two models

**public `addHasMany`** (*Phalcon\ Mvc\ ModelInterface* \$model, *mixed* \$fields, *string* \$referencedModel, *mixed* \$referencedFields, [*array* \$options])

Setup a relation 1-n between two models

**public `Phalcon\ Mvc\ Model\ Relation addHasManyToMany`** (*unknown* \$model, *string* \$fields, *string* \$intermediateModel, *string* \$intermediateFields, *string* \$intermediateReferencedFields, *string* \$referencedModel, *string* \$referencedFields, [*array* \$options])

Setups a relation n-m between two models

**public `boolean existsBelongsTo`** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has a belongsTo relation with another model

**public `boolean existsHasMany`** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has ahasMany relation with another model

**public `boolean existsHasOne`** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has ahasOne relation with another model

**public `boolean existsHasManyToMany`** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has ahasManyToMany relation with another model

**public `Phalcon\ Mvc\ Model\ Relation getRelationByAlias`** (*string* \$modelName, *string* \$alias)

Returns a relation by its alias

**public `Phalcon\ Mvc\ Model\ Resultset\ Simple getRelationRecords`** (*Phalcon\ Mvc\ Model\ Relation* \$relation, *string* \$method, *Phalcon\ Mvc\ ModelInterface* \$record, [*array* \$parameters])

Helper method to query records based on a relation definition

**public `object getReusableRecords`** (*string* \$modelName, *string* \$key)

Returns a reusable object from the internal list

**public `setReusableRecords`** (*string* \$modelName, *string* \$key, *mixed* \$records)

Stores a reusable record in the internal list

**public `clearReusableObjects`** ()

Clears the internal reusable list

**public `Phalcon\ Mvc\ Model\ ResultsetInterface getBelongsToRecords`** (*string* \$method, *string* \$modelName, *string* \$modelRelation, *Phalcon\ Mvc\ Model* \$record, [*array* \$parameters])

Gets belongsTo related records from a model

```
public Phalcon\ Mvc\ Model\ ResultsetInterface getHasManyRecords (string $method, string $modelName, string $modelRelation, Phalcon\ Mvc\ Model $record, [array $parameters])
```

GetshasMany related records from a model

```
public Phalcon\ Mvc\ Model\ ResultsetInterface getHasOneRecords (string $method, string $modelName, string $modelRelation, Phalcon\ Mvc\ Model $record, [array $parameters])
```

Gets belongsTo related records from a model

```
public Phalcon\ Mvc\ Model\ RelationInterface [] getBelongsTo (Phalcon\ Mvc\ ModelInterface $model)
```

Gets all the belongsTo relations defined in a model

```
<?php
```

```
$relations = $modelsManager->getBelongsTo(new Robots());
```

```
public Phalcon\ Mvc\ Model\ RelationInterface [] getHasMany (Phalcon\ Mvc\ ModelInterface $model)
```

GetshasMany relations defined on a model

```
public array getHasOne (Phalcon\ Mvc\ ModelInterface $model)
```

GetshasOne relations defined on a model

```
public Phalcon\ Mvc\ Model\ RelationInterface [] getHasManyToMany (Phalcon\ Mvc\ ModelInterface $model)
```

GetshasManyToMany relations defined on a model

```
public array getHasOneAndHasMany (Phalcon\ Mvc\ ModelInterface $model)
```

GetshasOne relations defined on a model

```
public Phalcon\ Mvc\ Model\ RelationInterface [] getRelations (string $modelName)
```

Query all the relationships defined on a model

```
public Phalcon\ Mvc\ Model\ RelationInterface getRelationsBetween (string $first, string $second)
```

Query the first relationship defined between two models

```
public Phalcon\ Mvc\ Model\ QueryInterface createQuery (string $phql)
```

Creates a Phalcon\ Mvc\ Model\ Query without execute it

```
public Phalcon\ Mvc\ Model\ QueryInterface executeQuery (string $phql, [array $placeholders])
```

Creates a Phalcon\ Mvc\ Model\ Query and execute it

```
public Phalcon\ Mvc\ Model\ Query\ BuilderInterface createBuilder ([string $params])
```

Creates a Phalcon\ Mvc\ Model\ Query\ Builder

```
public Phalcon\ Mvc\ Model\ QueryInterface getLastQuery ()
```

Returns the lastest query created or executed in the models manager

```
public registerNamespaceAlias (string $alias, string $namespace)
```

Registers shorter aliases for namespaces in PHQL statements

```
public string getNamespaceAlias (string $alias)
```

Returns a real namespace from its alias

```
public array getNamespaceAliases ()
```

Returns all the registered namespace aliases

`public __destruct ()`

Destroys the PHQL cache

## 2.54.164 Class Phalcon\Mvc\Model\Message

*implements Phalcon\ Mvc\ Model\ MessageInterface*

Encapsulates validation info generated before save/delete records fails

`<?php`

```
use Phalcon\Mvc\Model\Message as Message;

class Robots extends Phalcon\Mvc\Model
{

    public function beforeSave()
    {
        if ($this->name == 'Peter') {
            $text = "A robot cannot be named Peter";
            $field = "name";
            $type = "InvalidValue";
            $code = 103;
            $message = new Message($text, $field, $type, $code);
            $this->appendMessage($message);
        }
    }

}
```

### Methods

`public __construct (string $message, [string $field], [string $type])`

Phalcon\ Mvc\ Model\ Message constructor

`public Phalcon\ Mvc\ Model\ Message setType (string $type)`

Sets message type

`public string getType ()`

Returns message type

`public Phalcon\ Mvc\ Model\ Message setCode (string $code)`

Sets message code

`public string getCode ()`

Returns message code

`public Phalcon\ Mvc\ Model\ Message setMessage (string $message)`

Sets verbose message

`public string getMessage ()`

Returns verbose message

public *Phalcon\ Mvc\ Model\ Message* **setField** (*string* \$field)

Sets field name related to message

public *string* **getField** ()

Returns field name related to message

public *Phalcon\ Mvc\ Model\ Message* **setModel** (*Phalcon\ Mvc\ ModelInterface* \$model)

Set the model who generates the message

public *Phalcon\ Mvc\ ModelInterface* **getModel** ()

Returns the model that produced the message

public *string* **\_\_toString** ()

Magic \_\_toString method returns verbose message

public static *Phalcon\ Mvc\ Model\ Message* **\_\_set\_state** ([*unknown* \$properties])

Magic \_\_set\_state helps to re-build messages variable exporting

## 2.54.165 Abstract class Phalcon\ Mvc\ Model\ MetaData

*implements Phalcon\ DI\ InjectionAwareInterface, Phalcon\ Mvc\ Model\ MetaDataInterface*

Because Phalcon\ Mvc\ Model requires meta-data like field names, data types, primary keys, etc. this component collect them and store for further querying by Phalcon\ Mvc\ Model. Phalcon\ Mvc\ Model\ MetaData can also use adapters to store temporarily or permanently the meta-data. A standard Phalcon\ Mvc\ Model\ MetaData can be used to query model attributes:

<?php

```
$metaData = new Phalcon\ Mvc\ Model\ MetaData\ Memory();
$attributes = $metaData->getAttributes(new Robots());
print_r($attributes);
```

### Constants

*integer* MODELS\_ATTRIBUTES

*integer* MODELS\_PRIMARY\_KEY

*integer* MODELS\_NON\_PRIMARY\_KEY

*integer* MODELS\_NOT\_NULL

*integer* MODELS\_DATA\_TYPES

*integer* MODELS\_DATA\_TYPES\_NUMERIC

*integer* MODELS\_DATE\_AT

*integer* MODELS\_DATE\_IN

*integer* MODELS\_IDENTITY\_COLUMN

*integer* MODELS\_DATA\_TYPES\_BIND

*integer* MODELS\_AUTOMATIC\_DEFAULT\_INSERT

*integer* MODELS\_AUTOMATIC\_DEFAULT\_UPDATE

```
integer MODELS_COLUMN_MAP
integer MODELS_REVERSE_COLUMN_MAP
```

## Methods

protected `_initialize ()`

Initialize the metadata for certain table

public `setDI (Phalcon\DiInterface $dependencyInjector)`

Sets the DependencyInjector container

public `Phalcon\DiInterface getDI ()`

Returns the DependencyInjector container

public `setStrategy (Phalcon\Mvc\Model\MetaData\Strategy\Introspection $strategy)`

Set the meta-data extraction strategy

public `Phalcon\Mvc\Model\MetaData\Strategy\Introspection getStrategy ()`

Return the strategy to obtain the meta-data

public `array readMetaData (Phalcon\Mvc\ModelInterface $model)`

Reads the complete meta-data for certain model

`<?php`

```
print_r($metaData->readMetaData(new Robots()));
```

public `array readMetaDataIndex (Phalcon\Mvc\ModelInterface $model, int $index)`

Reads meta-data for certain model using a MODEL\_\* constant

`<?php`

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public `writeMetaDataIndex (Phalcon\Mvc\ModelInterface $model, int $index, mixed $data, unknown $replace)`

Writes meta-data for certain model using a MODEL\_\* constant

`<?php`

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public `array readColumnMap (Phalcon\Mvc\ModelInterface $model)`

Reads the ordered/reversed column map for certain model

`<?php`

```
print_r($metaData->readColumnMap(new Robots()));
```

public `readColumnMapIndex (Phalcon\Mvc\ModelInterface $model, int $index)`

Reads column-map information for certain model using a MODEL\_\* constant

```
<?php

print_r($metaData->readColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP));

public array getAttributes (Phalcon|Mvc\ModelInterface $model)
Returns table attributes names (fields)

<?php

print_r($metaData->getAttributes(new Robots()));

public array getPrimaryKeyAttributes (Phalcon|Mvc\ModelInterface $model)
Returns an array of fields which are part of the primary key

<?php

print_r($metaData->getPrimaryKeyAttributes(new Robots()));

public array getNonPrimaryKeyAttributes (Phalcon|Mvc\ModelInterface $model)
Returns an arrau of fields which are not part of the primary key

<?php

print_r($metaData->getNonPrimaryKeyAttributes(new Robots()));

public array getNotNullAttributes (Phalcon|Mvc\ModelInterface $model)
Returns an array of not null attributes

<?php

print_r($metaData->getNotNullAttributes(new Robots()));

public array getDataTypes (Phalcon|Mvc\ModelInterface $model)
Returns attributes and their data types

<?php

print_r($metaData->getDataTypes(new Robots()));

public array getDataTypesNumeric (Phalcon|Mvc\ModelInterface $model)
Returns attributes which types are numerical

<?php

print_r($metaData->getDataTypesNumeric(new Robots()));

public string getIdentityField (Phalcon|Mvc\ModelInterface $model)
Returns the name of identity field (if one is present)

<?php

print_r($metaData->getIdentityField(new Robots()));

public array getBindTypes (Phalcon|Mvc\ModelInterface $model)
Returns attributes and their bind data types
```

```
<?php

print_r($metaData->getBindTypes(new Robots()));

public array getAutomaticCreateAttributes (Phalcon\ Mvc\ ModelInterface $model)
>Returns attributes that must be ignored from the INSERT SQL generation

<?php

print_r($metaData->getAutomaticCreateAttributes(new Robots()));

public array getAutomaticUpdateAttributes (Phalcon\ Mvc\ ModelInterface $model)
>Returns attributes that must be ignored from the UPDATE SQL generation

<?php

print_r($metaData->getAutomaticUpdateAttributes(new Robots()));

public setAutomaticCreateAttributes (Phalcon\ Mvc\ ModelInterface $model, array $attributes,
unknown $replace)
>Set the attributes that must be ignored from the INSERT SQL generation

<?php

$metaData->setAutomaticCreateAttributes(new Robots(), array('created_at' => true));

public setAutomaticUpdateAttributes (Phalcon\ Mvc\ ModelInterface $model, array $attributes,
unknown $replace)
>Set the attributes that must be ignored from the UPDATE SQL generation

<?php

$metaData->setAutomaticUpdateAttributes(new Robots(), array('modified_at' => true));

public array getColumnMap (Phalcon\ Mvc\ ModelInterface $model)
>Returns the column map if any

<?php

print_r($metaData->getColumnMap(new Robots()));

public array getReverseColumnMap (Phalcon\ Mvc\ ModelInterface $model)
>Returns the reverse column map if any

<?php

print_r($metaData->getReverseColumnMap(new Robots()));

public boolean hasAttribute (Phalcon\ Mvc\ ModelInterface $model, string $attribute)
>Check if a model has certain attribute

<?php

var_dump($metaData->hasAttribute(new Robots(), 'name'));
```

public boolean isEmpty ()

Checks if the internal meta-data container is empty

<?php

```
var_dump($metaData->isEmpty());
```

public reset ()

Resets internal meta-data in order to regenerate it

<?php

```
$metaData->reset();
```

abstract public array read (string \$key) inherited from Phalcon\Mvc\Model\MetaDataInterface

Reads meta-data from the adapter

abstract public write (string \$key, array \$data) inherited from Phalcon\Mvc\Model\MetaDataInterface

Writes meta-data to the adapter

## 2.54.166 Class Phalcon\Mvc\Model\MetaData\Apc

extends abstract class Phalcon\ Mvc\ Model\ MetaData

implements Phalcon\ Mvc\ Model\ MetaDataInterface, Phalcon\ DI\ InjectionAwareInterface

Stores model meta-data in the APC cache. Data will be erased if the web server is restarted. By default meta-data is stored for 48 hours (172800 seconds). You can query the meta-data by printing apc\_fetch('PMM') or apc\_fetch('PMM\$my-app-id')

<?php

```
$metaData = new Phalcon\Mvc\Model\Metadata\Apc(array(
    'prefix' => 'my-app-id',
    'lifetime' => 86400
));
```

### Constants

integer MODELS\_ATTRIBUTES

integer MODELS\_PRIMARY\_KEY

integer MODELS\_NON\_PRIMARY\_KEY

integer MODELS\_NOT\_NULL

integer MODELS\_DATA\_TYPES

integer MODELS\_DATA\_TYPES\_NUMERIC

integer MODELS\_DATE\_AT

integer MODELS\_DATE\_IN

integer MODELS\_IDENTITY\_COLUMN

integer MODELS\_DATA\_TYPES\_BIND

```
integer MODELS_AUTOMATIC_DEFAULT_INSERT
integer MODELS_AUTOMATIC_DEFAULT_UPDATE
integer MODELS_COLUMN_MAP
integer MODELS_REVERSE_COLUMN_MAP
```

## Methods

**public \_\_construct ([array \$options])**

Phalcon\Mvc\Model\MetaData\Apc constructor

**public array read (string \$key)**

Reads meta-data from APC

**public write (string \$key, array \$data)**

Writes the meta-data to APC

**public reset ()**

...

**protected \_initialize ()** inherited from Phalcon\Mvc\Model\MetaData

Initialize the metadata for certain table

**public setDI (Phalcon\DiInterface \$dependencyInjector)** inherited from Phalcon\Mvc\Model\MetaData

Sets the DependencyInjector container

**public Phalcon\DiInterface getDI ()** inherited from Phalcon\Mvc\Model\MetaData

Returns the DependencyInjector container

**public setStrategy (Phalcon\Mvc\Model\MetaData\Strategy\Introspection \$strategy)** inherited from Phalcon\Mvc\Model\MetaData

Set the meta-data extraction strategy

**public Phalcon\Mvc\Model\MetaData\Strategy\Introspection getStrategy ()** inherited from Phalcon\Mvc\Model\MetaData

Return the strategy to obtain the meta-data

**public array readMetaData (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Reads the complete meta-data for certain model

<?php

```
print_r($metaData->readMetaData(new Robots()));
```

**public array readMetaDataIndex (Phalcon\Mvc\ModelInterface \$model, int \$index)** inherited from Phalcon\Mvc\Model\MetaData

Reads meta-data for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public **writeMetaDataIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, *int* \$index, *mixed* \$data, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Writes meta-data for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public *array* **readColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Reads the ordered/reversed column map for certain model

<?php

```
print_r($metaData->readColumnMap(new Robots()));
```

public **readColumnMapIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, *int* \$index) inherited from Phalcon\ Mvc\ Model\ MetaData

Reads column-map information for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->readColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP));
```

public *array* **getAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns table attributes names (fields)

<?php

```
print_r($metaData->getAttributes(new Robots()));
```

public *array* **getPrimaryKeyAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns an array of fields which are part of the primary key

<?php

```
print_r($metaData->getPrimaryKeyAttributes(new Robots()));
```

public *array* **getNonPrimaryKeyAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns an array of fields which are not part of the primary key

<?php

```
print_r($metaData->getNonPrimaryKeyAttributes(new Robots()));
```

public *array* **getNotNullAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns an array of not null attributes

<?php

```
print_r($metaData->getNotNullAttributes(new Robots()));
```

**public array getDataTypes (Phalcon\Mvc\ModelInterface \$model) inherited from Phalcon\Mvc\Model\MetaData**

Returns attributes and their data types

```
<?php
```

```
print_r($metaData->getDataTypes(new Robots()));
```

**public array getDataTypesNumeric (Phalcon\Mvc\ModelInterface \$model) inherited from Phalcon\Mvc\Model\MetaData**

Returns attributes which types are numerical

```
<?php
```

```
print_r($metaData->getDataTypesNumeric(new Robots()));
```

**public string getIdentityField (Phalcon\Mvc\ModelInterface \$model) inherited from Phalcon\Mvc\Model\MetaData**

Returns the name of identity field (if one is present)

```
<?php
```

```
print_r($metaData->getIdentityField(new Robots()));
```

**public array getBindTypes (Phalcon\Mvc\ModelInterface \$model) inherited from Phalcon\Mvc\Model\MetaData**

Returns attributes and their bind data types

```
<?php
```

```
print_r($metaData->getBindTypes(new Robots()));
```

**public array getAutomaticCreateAttributes (Phalcon\Mvc\ModelInterface \$model) inherited from Phalcon\Mvc\Model\MetaData**

Returns attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticCreateAttributes(new Robots()));
```

**public array getAutomaticUpdateAttributes (Phalcon\Mvc\ModelInterface \$model) inherited from Phalcon\Mvc\Model\MetaData**

Returns attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticUpdateAttributes(new Robots()));
```

**public setAutomaticCreateAttributes (Phalcon\Mvc\ModelInterface \$model, array \$attributes, unknown \$replace) inherited from Phalcon\Mvc\Model\MetaData**

Set the attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
$metaData->setAutomaticCreateAttributes(new Robots(), array('created_at' => true));
```

public **setAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the UPDATE SQL generation

<?php

```
$metaData->setAutomaticUpdateAttributes(new Robots(), array('modified_at' => true));
```

public *array* **getColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the column map if any

<?php

```
print_r($metaData->getColumnMap(new Robots()));
```

public *array* **getReverseColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the reverse column map if any

<?php

```
print_r($metaData->getReverseColumnMap(new Robots()));
```

public *boolean* **hasAttribute** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$attribute) inherited from Phalcon\ Mvc\ Model\ MetaData

Check if a model has certain attribute

<?php

```
var_dump($metaData->hasAttribute(new Robots(), 'name'));
```

public *boolean* **isEmpty** () inherited from Phalcon\ Mvc\ Model\ MetaData

Checks if the internal meta-data container is empty

<?php

```
var_dump($metaData->isEmpty());
```

## 2.54.167 Class Phalcon\ Mvc\ Model\ MetaData\ Files

*extends abstract class Phalcon\ Mvc\ Model\ MetaData*

*implements Phalcon\ Mvc\ Model\ MetaDataInterface, Phalcon\ DI\ InjectionAwareInterface*

Stores model meta-data in PHP files.

<?php

```
$metaData = new \Phalcon\ Mvc\ Model\ Metadata\ Files(array(
    'metaDataDir' => 'app/cache/metadata/',
));
```

## Constants

```
integer MODELS_ATTRIBUTES
integer MODELS_PRIMARY_KEY
integer MODELS_NON_PRIMARY_KEY
integer MODELS_NOT_NULL
integer MODELS_DATA_TYPES
integer MODELS_DATA_TYPES_NUMERIC
integer MODELS_DATE_AT
integer MODELS_DATE_IN
integer MODELS_IDENTITY_COLUMN
integer MODELS_DATA_TYPES_BIND
integer MODELS_AUTOMATIC_DEFAULT_INSERT
integer MODELS_AUTOMATIC_DEFAULT_UPDATE
integer MODELS_COLUMN_MAP
integer MODELS_REVERSE_COLUMN_MAP
```

## Methods

```
public __construct ([array $options])
Phalcon\Mvc\Model\MetaData\Files constructor

public array read (string $key)
Reads meta-data from files

public write (string $key, array $data)
Writes the meta-data to files

public reset ()
...

protected _initialize () inherited from Phalcon\Mvc\Model\MetaData
Initialize the metadata for certain table

public setDI (Phalcon\DiInterface $dependencyInjector) inherited from Phalcon\Mvc\Model\MetaData
Sets the DependencyInjector container

public Phalcon\DiInterface getDI () inherited from Phalcon\Mvc\Model\MetaData
Returns the DependencyInjector container

public Phalcon\Mvc\Model\MetaData\Strategy\Introspection $strategy) inherited from
Phalcon\Mvc\Model\MetaData
Set the meta-data extraction strategy

public Phalcon\Mvc\Model\MetaData\Strategy\Introspection getStrategy () inherited from
Phalcon\Mvc\Model\MetaData
```

Return the strategy to obtain the meta-data

```
public array readMetaData (Phalcon|Mvc\ModelInterface $model) inherited from  
Phalcon\Mvc\Model\MetaData
```

Reads the complete meta-data for certain model

```
<?php
```

```
print_r($metaData->readMetaData(new Robots()));
```

```
public array readMetaDataIndex (Phalcon|Mvc\ModelInterface $model, int $index) inherited from  
Phalcon\Mvc\Model\MetaData
```

Reads meta-data for certain model using a MODEL\_\* constant

```
<?php
```

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

```
public writeMetaDataIndex (Phalcon|Mvc\ModelInterface $model, int $index, mixed $data, unknown  
$replace) inherited from Phalcon\Mvc\Model\MetaData
```

Writes meta-data for certain model using a MODEL\_\* constant

```
<?php
```

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

```
public array readColumnMap (Phalcon|Mvc\ModelInterface $model) inherited from  
Phalcon\Mvc\Model\MetaData
```

Reads the ordered/reversed column map for certain model

```
<?php
```

```
print_r($metaData->readColumnMap(new Robots()));
```

```
public readColumnMapIndex (Phalcon|Mvc\ModelInterface $model, int $index) inherited from  
Phalcon\Mvc\Model\MetaData
```

Reads column-map information for certain model using a MODEL\_\* constant

```
<?php
```

```
print_r($metaData->readColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP));
```

```
public array getAttributes (Phalcon|Mvc\ModelInterface $model) inherited from  
Phalcon\Mvc\Model\MetaData
```

Returns table attributes names (fields)

```
<?php
```

```
print_r($metaData->getAttributes(new Robots()));
```

```
public array getPrimaryKeyAttributes (Phalcon|Mvc\ModelInterface $model) inherited from  
Phalcon\Mvc\Model\MetaData
```

Returns an array of fields which are part of the primary key

```
<?php
```

```
print_r($metaData->getPrimaryKeyAttributes(new Robots()));
```

**public array getNonPrimaryKeyAttributes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns an array of fields which are not part of the primary key

```
<?php
```

```
print_r($metaData->getNonPrimaryKeyAttributes(new Robots()));
```

**public array getNotNullAttributes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns an array of not null attributes

```
<?php
```

```
print_r($metaData->getNotNullAttributes(new Robots()));
```

**public array getDataTypes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes and their data types

```
<?php
```

```
print_r($metaData->getDataTypes(new Robots()));
```

**public array getDataTypesNumeric (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes which types are numerical

```
<?php
```

```
print_r($metaData->getDataTypesNumeric(new Robots()));
```

**public string getIdentityField (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns the name of identity field (if one is present)

```
<?php
```

```
print_r($metaData->getIdentityField(new Robots()));
```

**public array getBindTypes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes and their bind data types

```
<?php
```

```
print_r($metaData->getBindTypes(new Robots()));
```

**public array getAutomaticCreateAttributes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticCreateAttributes(new Robots()));
```

public *array* **getAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticUpdateAttributes(new Robots()));
```

public **setAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
$metaData->setAutomaticCreateAttributes(new Robots(), array('created_at' => true));
```

public **setAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
$metaData->setAutomaticUpdateAttributes(new Robots(), array('modified_at' => true));
```

public *array* **getColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the column map if any

```
<?php
```

```
print_r($metaData->getColumnMap(new Robots()));
```

public *array* **getReverseColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the reverse column map if any

```
<?php
```

```
print_r($metaData->getReverseColumnMap(new Robots()));
```

public *boolean* **hasAttribute** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$attribute) inherited from Phalcon\ Mvc\ Model\ MetaData

Check if a model has certain attribute

```
<?php
```

```
var_dump($metaData->hasAttribute(new Robots(), 'name'));
```

public *boolean* **isEmpty** () inherited from Phalcon\ Mvc\ Model\ MetaData

Checks if the internal meta-data container is empty

```
<?php
var_dump($metaData->isEmpty());
```

## 2.54.168 Class Phalcon\Mvc\Model\MetaData\Memory

*extends abstract class Phalcon\Mvc\Model\MetaData*

*implements Phalcon\Mvc\Model\MetaDataInterface, Phalcon\DI\InjectionAwareInterface*

Stores model meta-data in memory. Data will be erased when the request finishes

### Constants

```
integer MODELS_ATTRIBUTES
integer MODELS_PRIMARY_KEY
integer MODELS_NON_PRIMARY_KEY
integer MODELS_NOT_NULL
integer MODELS_DATA_TYPES
integer MODELS_DATA_TYPES_NUMERIC
integer MODELS_DATE_AT
integer MODELS_DATE_IN
integer MODELS_IDENTITY_COLUMN
integer MODELS_DATA_TYPES_BIND
integer MODELS_AUTOMATIC_DEFAULT_INSERT
integer MODELS_AUTOMATIC_DEFAULT_UPDATE
integer MODELS_COLUMN_MAP
integer MODELS_REVERSE_COLUMN_MAP
```

### Methods

public **\_\_construct** ([array \$options])

Phalcon\Mvc\Model\MetaData\Memory constructor

public array **read** (string \$key)

Reads the meta-data from temporal memory

public **write** (string \$key, unknown \$data)

Writes the meta-data to temporal memory

protected **\_initialize** () inherited from Phalcon\Mvc\Model\MetaData

Initialize the metadata for certain table

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\Mvc\Model\MetaData

Sets the DependencyInjector container

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\Mvc\Model\MetaData

Returns the DependencyInjector container

public **setStrategy** (*Phalcon\Mvc\Model\MetaData\Strategy\Introspection* \$strategy) inherited from Phalcon\Mvc\Model\MetaData

Set the meta-data extraction strategy

public *Phalcon\Mvc\Model\MetaData\Strategy\Introspection* **getStrategy** () inherited from Phalcon\Mvc\Model\MetaData

Return the strategy to obtain the meta-data

public array **readMetaData** (*Phalcon\Mvc\ModelInterface* \$model) inherited from Phalcon\Mvc\Model\MetaData

Reads the complete meta-data for certain model

<?php

```
print_r($metaData->readMetaData(new Robots()));
```

public array **readMetaDataIndex** (*Phalcon\Mvc\ModelInterface* \$model, int \$index) inherited from Phalcon\Mvc\Model\MetaData

Reads meta-data for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public **writeMetaDataIndex** (*Phalcon\Mvc\ModelInterface* \$model, int \$index, mixed \$data, unknown \$replace) inherited from Phalcon\Mvc\Model\MetaData

Writes meta-data for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public array **readColumnMap** (*Phalcon\Mvc\ModelInterface* \$model) inherited from Phalcon\Mvc\Model\MetaData

Reads the ordered/reversed column map for certain model

<?php

```
print_r($metaData->readColumnMap(new Robots()));
```

public **readColumnMapIndex** (*Phalcon\Mvc\ModelInterface* \$model, int \$index) inherited from Phalcon\Mvc\Model\MetaData

Reads column-map information for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->readColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP));
```

public array **getAttributes** (*Phalcon\Mvc\ModelInterface* \$model) inherited from Phalcon\Mvc\Model\MetaData

Returns table attributes names (fields)

```
<?php

print_r($metaData->getAttributes(new Robots()));

public array getPrimaryKeyAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns an array of fields which are part of the primary key

<?php

print_r($metaData->getPrimaryKeyAttributes(new Robots()));

public array getNonPrimaryKeyAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns an array of fields which are not part of the primary key

<?php

print_r($metaData->getNonPrimaryKeyAttributes(new Robots()));

public array getNotNullAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns an array of not null attributes

<?php

print_r($metaData->getNotNullAttributes(new Robots()));

public array getDataTypes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns attributes and their data types

<?php

print_r($metaData->getDataTypes(new Robots()));

public array getDataTypesNumeric (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns attributes which types are numerical

<?php

print_r($metaData->getDataTypesNumeric(new Robots()));

public string getIdentityField (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns the name of identity field (if one is present)

<?php

print_r($metaData->getIdentityField(new Robots()));

public array getBindTypes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns attributes and their bind data types
```

```
<?php
```

```
print_r($metaData->getBindTypes(new Robots()));
```

public *array* **getAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticCreateAttributes(new Robots()));
```

public *array* **getAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticUpdateAttributes(new Robots()));
```

public **setAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
$metaData->setAutomaticCreateAttributes(new Robots(), array('created_at' => true));
```

public **setAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
$metaData->setAutomaticUpdateAttributes(new Robots(), array('modified_at' => true));
```

public *array* **getColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the column map if any

```
<?php
```

```
print_r($metaData->getColumnMap(new Robots()));
```

public *array* **getReverseColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the reverse column map if any

```
<?php
```

```
print_r($metaData->getReverseColumnMap(new Robots()));
```

public *boolean* **hasAttribute** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$attribute) inherited from Phalcon\ Mvc\ Model\ MetaData

Check if a model has certain attribute

```
<?php

var_dump($metaData->hasAttribute(new Robots(), 'name'));

public boolean isEmpty () inherited from Phalcon\Mvc\Model\MetaData
Checks if the internal meta-data container is empty

<?php

var_dump($metaData->isEmpty());

public reset () inherited from Phalcon\Mvc\Model\MetaData
Resets internal meta-data in order to regenerate it

<?php

$metaData->reset();
```

## 2.54.169 Class Phalcon\Mvc\Model\MetaData\Session

*extends abstract class Phalcon\Mvc\Model\MetaData*

*implements Phalcon\Mvc\Model\MetaDataInterface, Phalcon\DI\InjectionAwareInterface*

Stores model meta-data in session. Data will be erased when the session finishes. Meta-data are permanent while the session is active. You can query the meta-data by printing `$_SESSION['$PMM$']`

```
<?php

$metaData = new Phalcon\Mvc\Model\MetaData\Session(array(
    'prefix' => 'my-app-id'
));
```

### Constants

```
integer MODELS_ATTRIBUTES
integer MODELS_PRIMARY_KEY
integer MODELS_NON_PRIMARY_KEY
integer MODELS_NOT_NULL
integer MODELS_DATA_TYPES
integer MODELS_DATA_TYPES_NUMERIC
integer MODELS_DATE_AT
integer MODELS_DATE_IN
integer MODELS_IDENTITY_COLUMN
integer MODELS_DATA_TYPES_BIND
integer MODELS_AUTOMATIC_DEFAULT_INSERT
integer MODELS_AUTOMATIC_DEFAULT_UPDATE
integer MODELS_COLUMN_MAP
```

*integer MODELS\_REVERSE\_COLUMN\_MAP*

## Methods

public **\_\_construct** ([array \$options])

Phalcon\Mvc\Model\MetaData\Session constructor

public **array read** (string \$key)

Reads meta-data from \$\_SESSION

public **write** (string \$key, array \$data)

Writes the meta-data to \$\_SESSION

public **reset** ()

...

protected **\_initialize** () inherited from Phalcon\Mvc\Model\MetaData

Initialize the metadata for certain table

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\Mvc\Model\MetaData

Sets the DependencyInjector container

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\Mvc\Model\MetaData

Returns the DependencyInjector container

public **setStrategy** (*Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection* \$strategy) inherited from Phalcon\Mvc\Model\MetaData

Set the meta-data extraction strategy

public *Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection* **getStrategy** () inherited from Phalcon\Mvc\Model\MetaData

Return the strategy to obtain the meta-data

public **array readMetaData** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\Mvc\Model\MetaData

Reads the complete meta-data for certain model

<?php

```
print_r($metaData->readMetaData(new Robots()));
```

public **array readMetaDataIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, int \$index) inherited from Phalcon\Mvc\Model\MetaData

Reads meta-data for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));
```

public **writeMetaDataIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, int \$index, mixed \$data, unknown \$replace) inherited from Phalcon\Mvc\Model\MetaData

Writes meta-data for certain model using a MODEL\_\* constant

```
<?php

print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name'));

public array readColumnMap (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Reads the ordered/reversed column map for certain model

<?php

print_r($metaData->readColumnMap(new Robots()));

public readColumnMapIndex (Phalcon\Mvc\ModelInterface $model, int $index) inherited from
Phalcon\Mvc\Model\MetaData

Reads column-map information for certain model using a MODEL_* constant

<?php

print_r($metaData->readColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP));

public array getAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns table attributes names (fields)

<?php

print_r($metaData->getAttributes(new Robots()));

public array getPrimaryKeyAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns an array of fields which are part of the primary key

<?php

print_r($metaData->getPrimaryKeyAttributes(new Robots()));

public array getNonPrimaryKeyAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns an arrau of fields which are not part of the primary key

<?php

print_r($metaData->getNonPrimaryKeyAttributes(new Robots()));

public array getNotNullAttributes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns an array of not null attributes

<?php

print_r($metaData->getNotNullAttributes(new Robots()));

public array getDataTypes (Phalcon\Mvc\ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData

Returns attributes and their data types
```

```
<?php
```

```
print_r($metaData->getDataTypes(new Robots()));
```

public array **getDataTypesNumeric** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes which types are numerical

```
<?php
```

```
print_r($metaData->getDataTypesNumeric(new Robots()));
```

public string **getIdentityField** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns the name of identity field (if one is present)

```
<?php
```

```
print_r($metaData->getIdentityField(new Robots()));
```

public array **getBindTypes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes and their bind data types

```
<?php
```

```
print_r($metaData->getBindTypes(new Robots()));
```

public array **getAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticCreateAttributes(new Robots()));
```

public array **getAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticUpdateAttributes(new Robots()));
```

public **setAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, array \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
$metaData->setAutomaticCreateAttributes(new Robots(), array('created_at' => true));
```

public **setAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, array \$attributes, *unknown* \$replace) inherited from Phalcon\ Mvc\ Model\ MetaData

Set the attributes that must be ignored from the UPDATE SQL generation

```
<?php

$metaData->setAutomaticUpdateAttributes(new Robots(), array('modified_at' => true));

public array getColumnMap (Phalcon\ Mvc\ ModelInterface $model) inherited from
Phalcon\ Mvc\ Model\ MetaData

Returns the column map if any

<?php

print_r($metaData->getColumnMap(new Robots()));

public array getReverseColumnMap (Phalcon\ Mvc\ ModelInterface $model) inherited from
Phalcon\ Mvc\ Model\ MetaData

Returns the reverse column map if any

<?php

print_r($metaData->getReverseColumnMap(new Robots()));

public boolean hasAttribute (Phalcon\ Mvc\ ModelInterface $model, string $attribute) inherited from
Phalcon\ Mvc\ Model\ MetaData

Check if a model has certain attribute

<?php

var_dump($metaData->hasAttribute(new Robots(), 'name'));

public boolean isEmpty () inherited from Phalcon\ Mvc\ Model\ MetaData

Checks if the internal meta-data container is empty

<?php

var_dump($metaData->isEmpty());
```

## 2.54.170 Class Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Annotations

Queries the table meta-data in order to introspect the model's metadata

### Methods

```
public array getMetaData (Phalcon\ Mvc\ ModelInterface $model, Phalcon\ DiInterface
$dependencyInjector)
```

The meta-data is obtained by reading the column descriptions from the database information schema

```
public array getColumnMaps ()
```

Read the model's column map, this can't be inferred

## 2.54.171 Class Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection

Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Instrospection Queries the table meta-data in order to introspect the model's metadata

## Methods

```
public array getMetaData (Phalcon\Mvc\ModelInterface $model, Phalcon\DiInterface $dependencyInjector)
```

The meta-data is obtained by reading the column descriptions from the database information schema

```
public array getColumnMaps (Phalcon\Mvc\ModelInterface $model, Phalcon\DiInterface $dependencyInjector)
```

Read the model's column map, this can't be inferred

### 2.54.172 Class Phalcon\Mvc\Model\MetaData\Xcache

*extends abstract class Phalcon\Mvc\Model\MetaData*

*implements Phalcon\Mvc\Model\MetaDataInterface, Phalcon\DI\InjectionAwareInterface*

Stores model meta-data in the XCache cache. Data will be erased if the web server is restarted. By default meta-data is stored for 48 hours (172800 seconds). You can query the meta-data by printing `xcache_get('{$PMM$}')` or `xcache_get('{$PMM$my-app-id}')`

```
<?php
```

```
$metaData = new Phalcon\Mvc\Model\MetaData\Xcache(array
    'prefix' => 'my-app-id',
    'lifetime' => 86400
));
```

## Constants

```
integer MODELS_ATTRIBUTES
integer MODELS_PRIMARY_KEY
integer MODELS_NON_PRIMARY_KEY
integer MODELS_NOT_NULL
integer MODELS_DATA_TYPES
integer MODELS_DATA_TYPES_NUMERIC
integer MODELS_DATE_AT
integer MODELS_DATE_IN
integer MODELS_IDENTITY_COLUMN
integer MODELS_DATA_TYPES_BIND
integer MODELS_AUTOMATIC_DEFAULT_INSERT
integer MODELS_AUTOMATIC_DEFAULT_UPDATE
integer MODELS_COLUMN_MAP
integer MODELS_REVERSE_COLUMN_MAP
```

## Methods

```
public __construct ([array $options])
Phalcon\Mvc\Model\MetaData\Xcache constructor

public array read (string $key)
Reads metadata from XCache

public write (string $key, array $data)
Writes the metadata to XCache

public reset ()
...

protected _initialize () inherited from Phalcon\Mvc\Model\MetaData
Initialize the metadata for certain table

public setDI (Phalcon\DiInterface $dependencyInjector) inherited from Phalcon\Mvc\Model\MetaData
Sets the DependencyInjector container

public Phalcon\DiInterface getDI () inherited from Phalcon\Mvc\Model\MetaData
Returns the DependencyInjector container

public setStrategy (Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection $strategy) inherited from
Phalcon\Mvc\Model\MetaData
Set the meta-data extraction strategy

public Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection getStrategy () inherited from
Phalcon\Mvc\Model\MetaData
Return the strategy to obtain the meta-data

public array readMetaData (Phalcon\ Mvc\ ModelInterface $model) inherited from
Phalcon\Mvc\Model\MetaData
Reads the complete meta-data for certain model

<?php

print_r($metaData->readMetaData(new Robots()));

public array readMetaDataIndex (Phalcon\ Mvc\ ModelInterface $model, int $index) inherited from
Phalcon\Mvc\Model\MetaData
Reads meta-data for certain model using a MODEL_* constant

<?php

print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')));

public writeMetaDataIndex (Phalcon\ Mvc\ ModelInterface $model, int $index, mixed $data, unknown
$replace) inherited from Phalcon\Mvc\Model\MetaData
Writes meta-data for certain model using a MODEL_* constant

<?php

print_r($metaData->writeColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP, array('leName' => 'name')))
```

public array **readColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Reads the ordered/reversed column map for certain model

<?php

```
print_r($metaData->readColumnMap(new Robots()));
```

public **readColumnMapIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, *int* \$index) inherited from Phalcon\ Mvc\ Model\ MetaData

Reads column-map information for certain model using a MODEL\_\* constant

<?php

```
print_r($metaData->readColumnMapIndex(new Robots(), MetaData::MODELS_REVERSE_COLUMN_MAP));
```

public array **getAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns table attributes names (fields)

<?php

```
print_r($metaData->getAttributes(new Robots()));
```

public array **getPrimaryKeyAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns an array of fields which are part of the primary key

<?php

```
print_r($metaData->getPrimaryKeyAttributes(new Robots()));
```

public array **getNonPrimaryKeyAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns an array of fields which are not part of the primary key

<?php

```
print_r($metaData->getNonPrimaryKeyAttributes(new Robots()));
```

public array **getNotNullAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns an array of not null attributes

<?php

```
print_r($metaData->getNotNullAttributes(new Robots()));
```

public array **getDataTypes** (*Phalcon\ Mvc\ ModelInterface* \$model) inherited from Phalcon\ Mvc\ Model\ MetaData

Returns attributes and their data types

<?php

```
print_r($metaData->getDataTypes(new Robots()));
```

**public array getDataTypesNumeric (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes which types are numerical

```
<?php
```

```
print_r($metaData->getDataTypesNumeric(new Robots()));
```

**public string getIdentityField (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns the name of identity field (if one is present)

```
<?php
```

```
print_r($metaData->getIdentityField(new Robots()));
```

**public array getBindTypes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes and their bind data types

```
<?php
```

```
print_r($metaData->getBindTypes(new Robots()));
```

**public array getAutomaticCreateAttributes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticCreateAttributes(new Robots()));
```

**public array getAutomaticUpdateAttributes (Phalcon\Mvc\ModelInterface \$model)** inherited from Phalcon\Mvc\Model\MetaData

Returns attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
print_r($metaData->getAutomaticUpdateAttributes(new Robots()));
```

**public setAutomaticCreateAttributes (Phalcon\Mvc\ModelInterface \$model, array \$attributes, unknown \$replace)** inherited from Phalcon\Mvc\Model\MetaData

Set the attributes that must be ignored from the INSERT SQL generation

```
<?php
```

```
$metaData->setAutomaticCreateAttributes(new Robots(), array('created_at' => true));
```

**public setAutomaticUpdateAttributes (Phalcon\Mvc\ModelInterface \$model, array \$attributes, unknown \$replace)** inherited from Phalcon\Mvc\Model\MetaData

Set the attributes that must be ignored from the UPDATE SQL generation

```
<?php
```

```
$metaData->setAutomaticUpdateAttributes(new Robots(), array('modified_at' => true));
```

public array **getColumnMap** (*Phalcon|Mvc\ModelInterface* \$model) inherited from Phalcon\Mvc\Model\MetaData

Returns the column map if any

<?php

```
print_r($metaData->getColumnMap(new Robots()));
```

public array **getReverseColumnMap** (*Phalcon|Mvc\ModelInterface* \$model) inherited from Phalcon\Mvc\Model\MetaData

Returns the reverse column map if any

<?php

```
print_r($metaData->getReverseColumnMap(new Robots()));
```

public boolean **hasAttribute** (*Phalcon|Mvc\ModelInterface* \$model, *string* \$attribute) inherited from Phalcon\Mvc\Model\MetaData

Check if a model has certain attribute

<?php

```
var_dump($metaData->hasAttribute(new Robots(), 'name'));
```

public boolean **isEmpty** () inherited from Phalcon\Mvc\Model\MetaData

Checks if the internal meta-data container is empty

<?php

```
var_dump($metaData->isEmpty());
```

## 2.54.173 Class Phalcon\Mvc\Model\Query

implements *Phalcon|Mvc\Model\QueryInterface*, *Phalcon|DI\InjectionAwareInterface*

This class takes a PHQL intermediate representation and executes it.

<?php

```
$phql = "SELECT c.price*0.16 AS taxes, c.* FROM Cars AS c JOIN Brands AS b
        WHERE b.name = :name: ORDER BY c.name";

$result = $manager->executeQuery($phql, array(
    'name' => 'Lamborghini'
));

foreach ($result as $row) {
    echo "Name: ", $row->cars->name, "\n";
    echo "Price: ", $row->cars->price, "\n";
    echo "Taxes: ", $row->taxes, "\n";
}
```

### Constants

integer **TYPE\_SELECT**

```
integer TYPE_INSERT  
integer TYPE_UPDATE  
integer TYPE_DELETE
```

## Methods

```
public __construct (string $phql)  
Phalcon\Mvc\Model\Query constructor  
public setDI (Phalcon\DiInterface $dependencyInjector)  
Sets the dependency injection container  
public Phalcon\DiInterface getDI ()  
Returns the dependency injection container  
public Phalcon\Mvc\Model\Query setUniqueRow (boolean $uniqueRow)  
Tells to the query if only the first row in the resultset must be returned  
public boolean getUniqueRow ()  
Check if the query is programmed to get only the first row in the resultset  
protected string _getQualified ()  
Replaces the model's name to its source name in a qualified-name expression  
protected string _getCallArgument ()  
Resolves a expression in a single call argument  
protected string _getFunctionCall ()  
Resolves a expression in a single call argument  
protected string _getExpression ()  
Resolves an expression from its intermediate code into a string  
protected array _getSelectColumn ()  
Resolves a column from its intermediate representation into an array used to determine if the resulset produced is simple or complex  
protected string _getTable ()  
Resolves a table in a SELECT statement checking if the model exists  
protected array _getJoin ()  
Resolves a JOIN clause checking if the associated models exist  
protected string _getJoinType ()  
Resolves a JOIN type  
protected array _getSingleJoin ()  
Resolves joins involving has-one/belongs-to/has-many relations  
protected array _getMultiJoin ()  
Resolves joins involving many-to-many relations
```

protected *array* **\_getJoins ()**

Processes the JOINS in the query returning an internal representation for the database dialect

protected *string* **\_getOrderClause ()**

Returns a processed order clause for a SELECT statement

protected *string* **\_getGroupClause ()**

Returns a processed group clause for a SELECT statement

protected **\_getLimitClause ()**

...

protected *array* **\_prepareSelect ()**

Analyzes a SELECT intermediate code and produces an array to be executed later

protected *array* **\_prepareInsert ()**

Analyzes an INSERT intermediate code and produces an array to be executed later

protected *array* **\_prepareUpdate ()**

Analyzes an UPDATE intermediate code and produces an array to be executed later

protected *array* **\_prepareDelete ()**

Analyzes a DELETE intermediate code and produces an array to be executed later

public *array* **parse ()**

Parses the intermediate code produced by Phalcon\Mvc\Model\Query\Lang generating another intermediate representation that could be executed by Phalcon\Mvc\Model\Query

public *Phalcon\ Mvc\ Model\ Query* **cache (array \$cacheOptions)**

Sets the cache parameters of the query

public **getCacheOptions ()**

Returns the current cache options

public *Phalcon\ Cache\ BackendInterface* **getCache ()**

Returns the current cache backend instance

protected *Phalcon\ Mvc\ Model\ ResultsetInterface* **\_executeSelect ()**

Executes the SELECT intermediate representation producing a Phalcon\Mvc\Model\Resultset

protected *Phalcon\ Mvc\ Model\ Query\ StatusInterface* **\_executeInsert ()**

Executes the INSERT intermediate representation producing a Phalcon\Mvc\Model\Query>Status

protected *Phalcon\ Mvc\ Model\ ResultsetInterface* **\_getRelatedRecords ()**

Query the records on which the UPDATE/DELETE operation well be done

protected *Phalcon\ Mvc\ Model\ Query\ StatusInterface* **\_executeUpdate ()**

Executes the UPDATE intermediate representation producing a Phalcon\Mvc\Model\Query>Status

protected *Phalcon\ Mvc\ Model\ Query\ StatusInterface* **\_executeDelete ()**

Executes the DELETE intermediate representation producing a Phalcon\Mvc\Model\Query>Status

public *mixed* **execute ([array \$bindParams], [array \$bindTypes])**

Executes a parsed PHQL statement

`public halconMvcModelInterface getSingleResult ([array $bindParams], [array $bindTypes])`

Executes the query returning the first result

`public Phalcon|Mvc|Model|Query setType (int $type)`

Sets the type of PHQL statement to be executed

`public int getType ()`

Gets the type of PHQL statement executed

`public Phalcon|Mvc|Model|Query setBindParams (array $bindParams)`

Set default bind parameters

`public array getBindParams ()`

Returns default bind params

`public Phalcon|Mvc|Model|Query setBindTypes (array $bindTypes)`

Set default bind parameters

`public array getBindTypes ()`

Returns default bind types

`public Phalcon|Mvc|Model|Query setIntermediate (array $intermediate)`

Allows to set the IR to be executed

`public array getIntermediate ()`

Returns the intermediate representation of the PHQL statement

## 2.54.174 Class Phalcon\ Mvc\ Model\ Query\ Builder

*implements Phalcon|Mvc|Model|Query|BuilderInterface, Phalcon|DI|InjectionAwareInterface*

Helps to create PHQL queries using an OO interface

`<?php`

```
$resultset = $this->modelsManager->createBuilder()
    ->from('Robots')
    ->join('RobotsParts')
    ->limit(20)
    ->orderBy('Robots.name')
    ->getQuery()
    ->execute();
```

### Methods

`public __construct ([array $params])`

Phalcon\ Mvc\ Model\ Query\ Builder constructor

&lt;?php

```
$params = array(
    'models'      => array('Users'),
    'columns'     => array('id', 'name', 'status'),
    'conditions'  => array(
        array(
            "created > :min: AND created < :max:",
            array("min" => '2013-01-01', 'max' => '2014-01-01'),
            array("min" => PDO::PARAM_STR, 'max' => PDO::PARAM_STR),
        ),
    ),
    // or 'conditions' => "created > '2013-01-01' AND created < '2014-01-01'",
    'group'       => array('id', 'name'),
    'having'      => "name = 'Kamil'",
    'order'        => array('name', 'id'),
    'limit'        => 20,
    'offset'       => 20,
    // or 'limit' => array(20, 20),
);
$queryBuilder = new Phalcon\Mvc\Model\Query\Builder($params);
```

public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **distinct** (*unknown* \$distinct)

Sets SELECT DISTINCT / SELECT ALL flag

public *bool* **getDistinct** ()

Returns SELECT DISTINCT / SELECT ALL flag

public *Phalcon\ Mvc\ Model\ Query\ Builder* **setDI** (*Phalcon\ DiInterface* \$dependencyInjector)

Sets the DependencyInjector container

public *Phalcon\ DiInterface* **getDI** ()

Returns the DependencyInjector container

public *Phalcon\ Mvc\ Model\ Query\ Builder* **columns** (*string/array* \$columns)

Sets the columns to be queried

&lt;?php

```
$builder->columns(array('id', 'name'));
```

public *string/array* **getColumns** ()

Return the columns to be queried

public *Phalcon\ Mvc\ Model\ Query\ Builder* **from** (*string/array* \$models)

Sets the models who makes part of the query

&lt;?php

```
$builder->from('Robots');
$builder->from(array('Robots', 'RobotsParts'));
```

public *Phalcon\ Mvc\ Model\ Query\ Builder* **addFrom** (*string* \$model, [*string* \$alias])

Add a model to take part of the query

```
<?php
```

```
$builder->addFrom('Robots', 'r');
```

**public string/array getFrom ()**

Return the models who makes part of the query

```
public Phalcon\ Mvc\ Model\ Query\ Builder join (string $model, [string $conditions], [string $alias])
```

Adds a join to the query

```
<?php
```

```
$builder->join('Robots');
$builder->join('Robots', 'r.id = RobotsParts.robots_id');
$builder->join('Robots', 'r.id = RobotsParts.robots_id', 'r');
$builder->join('Robots', 'r.id = RobotsParts.robots_id', 'r', 'LEFT');
```

```
public Phalcon\ Mvc\ Model\ Query\ Builder innerJoin (string $model, [string $conditions], [string $alias])
```

Adds a INNER join to the query

```
<?php
```

```
$builder->innerJoin('Robots');
$builder->innerJoin('Robots', 'r.id = RobotsParts.robots_id');
$builder->innerJoin('Robots', 'r.id = RobotsParts.robots_id', 'r');
```

```
public Phalcon\ Mvc\ Model\ Query\ Builder leftJoin (string $model, [string $conditions], [string $alias])
```

Adds a LEFT join to the query

```
<?php
```

```
$builder->leftJoin('Robots', 'r.id = RobotsParts.robots_id', 'r');
```

```
public Phalcon\ Mvc\ Model\ Query\ Builder rightJoin (string $model, [string $conditions], [string $alias])
```

Adds a RIGHT join to the query

```
<?php
```

```
$builder->rightJoin('Robots', 'r.id = RobotsParts.robots_id', 'r');
```

```
public Phalcon\ Mvc\ Model\ Query\ Builder where (string $conditions, [array $bindParams], [array $bindTypes])
```

Sets the query conditions

```
<?php
```

```
$builder->where('name = "Peter"');
$builder->where('name = :name: AND id > :id:', array('name' => 'Peter', 'id' => 100));
```

```
public Phalcon\ Mvc\ Model\ Query\ Builder andWhere (string $conditions, [array $bindParams], [array $bindTypes])
```

Appends a condition to the current conditions using a AND operator

```
<?php
```

```
$builder->andWhere('name = "Peter"');
$builder->andWhere('name = :name: AND id > :id:', array('name' => 'Peter', 'id' => 100));
```

public *Phalcon|Mvc|Model|Query\Builder* **orWhere** (*string \$conditions, [array \$bindParams, [array \$bindTypes]]*)

Appends a condition to the current conditions using a OR operator

```
<?php
```

```
$builder->orWhere('name = "Peter"');
$builder->orWhere('name = :name: AND id > :id:', array('name' => 'Peter', 'id' => 100));
```

public *Phalcon|Mvc|Model|Query\Builder* **betweenWhere** (*string \$expr, mixed \$minimum, mixed \$maximum*)

Appends a BETWEEN condition to the current conditions

```
<?php
```

```
$builder->betweenWhere('price', 100.25, 200.50);
```

public *Phalcon|Mvc|Model|Query\Builder* **notBetweenWhere** (*string \$expr, mixed \$minimum, mixed \$maximum*)

Appends a NOT BETWEEN condition to the current conditions

```
<?php
```

```
$builder->notBetweenWhere('price', 100.25, 200.50);
```

public *Phalcon|Mvc|Model|Query\Builder* **inWhere** (*string \$expr, array \$values*)

Appends an IN condition to the current conditions

```
<?php
```

```
$builder->inWhere('id', [1, 2, 3]);
```

public *Phalcon|Mvc|Model|Query\Builder* **notInWhere** (*string \$expr, array \$values*)

Appends a NOT IN condition to the current conditions

```
<?php
```

```
$builder->notInWhere('id', [1, 2, 3]);
```

public *string/array getWhere ()*

Return the conditions for the query

public *Phalcon|Mvc|Model|Query\Builder orderBy* (*string \$orderBy*)

Sets a ORDER BY condition clause

```
<?php
```

```
$builder->orderBy('Robots.name');
$builder->orderBy(array('1', 'Robots.name'));
```

public *string/array getOrderBy ()*

Returns the set ORDER BY clause

```
public Phalcon\ Mvc\ Model\ Query\ Builder having (string $having)
Sets a HAVING condition clause. You need to escape PHQL reserved words using [ and ] delimiters
<?php

$builder->having('SUM(Robots.price) > 0');

public string/array getHaving ()
Return the current having clause

public Phalcon\ Mvc\ Model\ Query\ Builder limit (int $limit, [int $offset])
Sets a LIMIT clause, optionally a offset clause
<?php

$builder->limit(100);
$builder->limit(100, 20);

public string/array getLimit ()
Returns the current LIMIT clause

public Phalcon\ Mvc\ Model\ Query\ Builder offset (int $offset)
Sets an OFFSET clause
<?php

$builder->offset(30);

public string/array getOffset ()
Returns the current OFFSET clause

public Phalcon\ Mvc\ Model\ Query\ Builder groupBy (string $group)
Sets a GROUP BY clause
<?php

$builder->groupBy(array('Robots.name'));

public string getGroupBy ()
Returns the GROUP BY clause

public string getPhql ()
Returns a PHQL statement built based on the builder parameters

public Phalcon\ Mvc\ Model\ Query\ getQuery ()
Returns the query built
```

## 2.54.175 Abstract class Phalcon\ Mvc\ Model\ Query\ Lang

PHQL is implemented as a parser (written in C) that translates syntax in that of the target RDBMS. It allows Phalcon to offer a unified SQL language to the developer, while internally doing all the work of translating PHQL instructions to the most optimal SQL instructions depending on the RDBMS type associated with a model. To achieve the highest performance possible, we wrote a parser that uses the same technology as

SQLite. This technology provides a small in-memory parser with a very low memory footprint that is also thread-safe.

```
<?php
```

```
$intermediate = Phalcon\Mvc\Model\Query\Lang::parsePHQL("SELECT r.* FROM Robots r LIMIT 10");
```

## Methods

```
public static string parsePHQL (string $phql)
```

Parses a PHQL statement returning an intermediate representation (IR)

## 2.54.176 Class Phalcon\ Mvc\ Model\ Query\ Status

*implements Phalcon\ Mvc\ Model\ Query\ StatusInterface*

This class represents the status returned by a PHQL statement like INSERT, UPDATE or DELETE. It offers context information and the related messages produced by the model which finally executes the operations when it fails

```
<?php
```

```
$phql = "UPDATE Robots SET name = :name:, type = :type:, year = :year: WHERE id = :id:";  
$status = $app->modelsManager->executeQuery($phql, array(  
    'id' => 100,  
    'name' => 'Astroy Boy',  
    'type' => 'mechanical',  
    'year' => 1959  
));  
  
//Check if the update was successful  
if ($status->success() == true) {  
    echo 'OK';  
}
```

## Methods

```
public __construct (boolean $success, Phalcon\ Mvc\ ModelInterface $model)
```

```
public Phalcon\ Mvc\ ModelInterface getModel ()
```

Returns the model that executed the action

```
public Phalcon\ Mvc\ Model\ MessageInterface [] getMessages ()
```

Returns the messages produced by a failed operation

```
public boolean success ()
```

Allows to check if the executed operation was successful

## 2.54.177 Class Phalcon\ Mvc\ Model\ Relation

*implements Phalcon\ Mvc\ Model\ RelationInterface*

This class represents a relationship between two models

## Constants

```
integer BELONGS_TO
integer HAS_ONE
integer HAS_MANY
integer HAS_ONE_THROUGH
integer HAS_MANY_THROUGH
integer NO_ACTION
integer ACTION_RESTRICT
integer ACTION.Cascade
```

## Methods

**public \_\_construct** (*int* \$type, *string* \$referencedModel, *string/array* \$fields, *string/array* \$referencedFields, [*array* \$options])

Phalcon\Mvc\Model\Relation constructor

**public setIntermediateRelation** (*string/array* \$intermediateFields, *string* \$intermediateModel, *string* \$intermediateReferencedFields)

Sets the intermediate model data for has-\*-through relations

**public int getType ()**

Returns the relation type

**public string getReferencedModel ()**

Returns the referenced model

**public string/array getFields ()**

Returns the fields

**public string/array getReferencedFields ()**

Returns the referenced fields

**public string/array getOptions ()**

Returns the options

**public string/array isForeignKey ()**

Check whether the relation act as a foreign key

**public string/array getForeignKey ()**

Returns the foreign key configuration

**public boolean isThrough ()**

Check whether the relation is a ‘many-to-many’ relation or not

**public boolean isReusable ()**

Check if records returned by getting belongs-to/has-many are implicitly cached during the current request

**public string/array getIntermediateFields ()**

Gets the intermediate fields for has-\* -through relations

`public string getIntermediateModel ()`

Gets the intermediate model for has-\* -through relations

`public string/array getIntermediateReferencedFields ()`

Gets the intermediate referenced fields for has-\* -through relations

## 2.54.178 Abstract class Phalcon\ Mvc\ Model\ Resultset

*implements Phalcon\ Mvc\ Model\ ResultsetInterface, Iterator, Traversable, SeekableIterator, Countable, ArrayAccess, Serializable*

This component allows to Phalcon\ Mvc\ Model returns large resultsets with the minimum memory consumption. Resultsets can be traversed using a standard foreach or a while statement. If a resultset is serialized it will dump all the rows into a big array. Then unserialize will retrieve the rows as they were before serializing.

```
<?php

//Using a standard foreach
$robots = Robots::find(array("type='virtual'", "order" => "name"));
foreach ($robots as $robot) {
    echo $robot->name, "\n";
}

//Using a while
$robots = Robots::find(array("type='virtual'", "order" => "name"));
$robots->rewind();
while ($robots->valid()) {
    $robot = $robots->current();
    echo $robot->name, "\n";
    $robots->next();
}
```

### Constants

`integer TYPE_RESULT_FULL`

`integer TYPE_RESULT_PARTIAL`

`integer HYDRATE_RECORDS`

`integer HYDRATE_OBJECTS`

`integer HYDRATE_ARRAYS`

### Methods

`public next ()`

Moves cursor to next row in the resultset

`public int key ()`

Gets pointer number of active row in the resultset

`public rewind ()`

Rewinds resultset to its beginning

public **seek** (*int \$position*)

Changes internal pointer to a specific position in the resultset

public *int count ()*

Counts how many rows are in the resultset

public *boolean offsetExists (unknown \$property)*

Checks whether offset exists in the resultset

public *Phalcon\ Mvc\ ModelInterface offsetGet (unknown \$property)*

Gets row in a specific position of the resultset

public *offsetSet (unknown \$property, Phalcon\ Mvc\ ModelInterface \$value)*

Resultsets cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

public **offsetUnset** (*unknown \$property*)

Resultsets cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

public *int getType ()*

Returns the internal type of data retrieval that the resultset is using

public *Phalcon\ Mvc\ ModelInterface getFirst ()*

Get first row in the resultset

public *Phalcon\ Mvc\ ModelInterface getLast ()*

Get last row in the resultset

public *Phalcon\ Mvc\ Model\ Resultset setIsFresh (boolean \$isFresh)*

Set if the resultset is fresh or an old one cached

public *boolean isFresh ()*

Tell if the resultset is fresh or an old one cached

public *Phalcon\ Mvc\ Model\ Resultset setHydrateMode (int \$hydrateMode)*

Sets the hydration mode in the resultset

public *int getHydrateMode ()*

Returns the current hydration mode

public *Phalcon\ Cache\ BackendInterface getCache ()*

Returns the associated cache for the resultset

public *Phalcon\ Mvc\ ModelInterface current ()*

Returns current row in the resultset

public *Phalcon\ Mvc\ Model\ MessageInterface [] getMessages ()*

Returns the error messages produced by a batch operation

public *boolean delete ([Closure \$conditionCallback])*

Deletes every record in the resultset

public *Phalcon\ Mvc\ Model* [] **filter** (*callback \$filter*)

Filters a resultset returning only those the developer requires

<?php

```
$filtered = $robots->filter(function($robot){  
    if ($robot->id < 3) {  
        return $robot;  
    }  
});
```

abstract public *array toArray* () inherited from *Phalcon\ Mvc\ Model\ ResultsetInterface*

Returns a complete resultset as an array, if the resultset has a big number of rows it could consume more memory than currently it does.

abstract public **valid** () inherited from *Iterator*

...

abstract public **serialize** () inherited from *Serializable*

...

abstract public **unserialize** (*unknown \$serialized*) inherited from *Serializable*

...

## 2.54.179 Class *Phalcon\ Mvc\ Model\ Resultset\ Complex*

*extends abstract class Phalcon\ Mvc\ Model\ Resultset*

*implements Serializable, ArrayAccess, Countable, SeekableIterator, Traversable, Iterator,*  
*Phalcon\ Mvc\ Model\ ResultsetInterface*

Complex resultsets may include complete objects and scalar values. This class builds every complex row as it is required

### Constants

*integer TYPE\_RESULT\_FULL*

*integer TYPE\_RESULT\_PARTIAL*

*integer HYDRATE\_RECORDS*

*integer HYDRATE\_OBJECTS*

*integer HYDRATE\_ARRAYS*

### Methods

public **\_\_construct** (*array \$columnsTypes, Phalcon\ Db\ ResultInterface \$result, [Phalcon\ Cache\ BackendInterface \$cache]*)

*Phalcon\ Mvc\ Model\ Resultset\ Complex* constructor

public *boolean valid* ()

Check whether internal resource has rows to fetch

**public array toArray ()**

Returns a complete resultset as an array, if the resultset has a big number of rows it could consume more memory than currently it does.

**public string serialize ()**

Serializing a resultset will dump all related rows into a big array

**public unserialize ([*unknown* \$serialized])**

Unserializing a resultset will allow to only works on the rows present in the saved state

**public next ()** inherited from Phalcon\Mvc\Model\Resultset

Moves cursor to next row in the resultset

**public int key ()** inherited from Phalcon\Mvc\Model\Resultset

Gets pointer number of active row in the resultset

**public rewind ()** inherited from Phalcon\Mvc\Model\Resultset

Rewinds resultset to its beginning

**public seek (*int* \$position)** inherited from Phalcon\Mvc\Model\Resultset

Changes internal pointer to a specific position in the resultset

**public int count ()** inherited from Phalcon\Mvc\Model\Resultset

Counts how many rows are in the resultset

**public boolean offsetExists (*unknown* \$property)** inherited from Phalcon\Mvc\Model\Resultset

Checks whether offset exists in the resultset

**public *Phalcon\Mvc\ModelInterface* offsetGet (*unknown* \$property)** inherited from Phalcon\Mvc\Model\Resultset

Gets row in a specific position of the resultset

**public offsetSet (*unknown* \$property, *Phalcon\Mvc\ModelInterface* \$value)** inherited from Phalcon\Mvc\Model\Resultset

Resultsets cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

**public offsetUnset (*unknown* \$property)** inherited from Phalcon\Mvc\Model\Resultset

Resultsets cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

**public int getType ()** inherited from Phalcon\Mvc\Model\Resultset

Returns the internal type of data retrieval that the resultset is using

**public *Phalcon\Mvc\ModelInterface* getFirst ()** inherited from Phalcon\Mvc\Model\Resultset

Get first row in the resultset

**public *Phalcon\Mvc\ModelInterface* getLast ()** inherited from Phalcon\Mvc\Model\Resultset

Get last row in the resultset

**public *Phalcon\Mvc\Model\Resultset* setIsFresh (*boolean* \$isFresh)** inherited from Phalcon\Mvc\Model\Resultset

Set if the resultset is fresh or an old one cached

public *boolean* **isFresh** () inherited from Phalcon\Mvc\Model\Resultset

Tell if the resultset is fresh or an old one cached

public *Phalcon\ Mvc\ Model\ Resultset* **setHydrateMode** (*int* \$hydrateMode) inherited from Phalcon\Mvc\Model\Resultset

Sets the hydration mode in the resultset

public *int* **getHydrateMode** () inherited from Phalcon\Mvc\Model\Resultset

Returns the current hydration mode

public *Phalcon\ Cache\ BackendInterface* **getCache** () inherited from Phalcon\Mvc\Model\Resultset

Returns the associated cache for the resultset

public *Phalcon\ Mvc\ ModelInterface* **current** () inherited from Phalcon\Mvc\Model\Resultset

Returns current row in the resultset

public *Phalcon\ Mvc\ Model\ MessageInterface* [] **getMessages** () inherited from Phalcon\Mvc\Model\Resultset

Returns the error messages produced by a batch operation

public *boolean* **delete** ([*Closure* \$conditionCallback]) inherited from Phalcon\Mvc\Model\Resultset

Deletes every record in the resultset

public *Phalcon\ Mvc\ Model* [] **filter** (*callback* \$filter) inherited from Phalcon\Mvc\Model\Resultset

Filters a resultset returning only those the developer requires

<?php

```
$filtered = $robots->filter(function($robot){  
    if ($robot->id < 3) {  
        return $robot;  
    }  
});
```

## 2.54.180 Class Phalcon\Mvc\Model\Resultset\Simple

*extends* abstract class *Phalcon\ Mvc\ Model\ Resultset*

*implements* Serializable, ArrayAccess, Countable, SeekableIterator, Traversable, Iterator, *Phalcon\ Mvc\ Model\ ResultsetInterface*

Simple resultsets only contains complete objects. This class builds every complete object as it is required

### Constants

*integer* **TYPE\_RESULT\_FULL**

*integer* **TYPE\_RESULT\_PARTIAL**

*integer* **HYDRATE\_RECORDS**

*integer* **HYDRATE\_OBJECTS**

*integer* **HYDRATE\_ARRAYS**

## Methods

**public \_\_construct (array \$columnMap, *Phalcon\ Mvc\ ModelInterface* \$model, *Phalcon\ Db\ Result\ Pdo* \$result, [*Phalcon\ Cache\ BackendInterface* \$cache], [boolean \$keepSnapshots])**

*Phalcon\ Mvc\ Model\ Resultset\ Simple* constructor

**public boolean valid ()**

Check whether the internal resource has rows to fetch

**public array toArray ([boolean \$renameColumns])**

Returns a complete resultset as an array, if the resultset has a big number of rows it could consume more memory than it currently does. Exporting the resultset to an array couldn't be faster with a large number of records

**public string serialize ()**

Serializing a resultset will dump all related rows into a big array

**public unserialize ([unknown \$serialized])**

Unserializing a resultset only works on the rows present in the saved state

**public next ()** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Moves cursor to next row in the resultset

**public int key ()** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Gets pointer number of active row in the resultset

**public rewind ()** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Rewinds resultset to its beginning

**public seek (int \$position)** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Changes internal pointer to a specific position in the resultset

**public int count ()** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Counts how many rows are in the resultset

**public boolean offsetExists (unknown \$property)** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Checks whether offset exists in the resultset

**public *Phalcon\ Mvc\ ModelInterface* offsetGet (unknown \$property)** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Gets row in a specific position of the resultset

**public offsetSet (unknown \$property, *Phalcon\ Mvc\ ModelInterface* \$value)** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Resultsets cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

**public offsetUnset (unknown \$property)** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Resultsets cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

**public int getType ()** inherited from *Phalcon\ Mvc\ Model\ Resultset*

Returns the internal type of data retrieval that the resultset is using

public *Phalcon\ Mvc\ ModelInterface* **getFirst** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Get first row in the resultset

public *Phalcon\ Mvc\ ModelInterface* **getLast** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Get last row in the resultset

public *Phalcon\ Mvc\ Model\ Resultset* **setIsFresh** (*boolean* \$isFresh) inherited from Phalcon\ Mvc\ Model\ Resultset  
Set if the resultset is fresh or an old one cached

public *boolean* **isFresh** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Tell if the resultset is fresh or an old one cached

public *Phalcon\ Mvc\ Model\ Resultset* **setHydrateMode** (*int* \$hydrateMode) inherited from Phalcon\ Mvc\ Model\ Resultset  
Sets the hydration mode in the resultset

public *int* **getHydrateMode** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Returns the current hydration mode

public *Phalcon\ Cache\ BackendInterface* **getCache** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Returns the associated cache for the resultset

public *Phalcon\ Mvc\ ModelInterface* **current** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Returns current row in the resultset

public *Phalcon\ Mvc\ Model\ MessageInterface* [] **getMessages** () inherited from Phalcon\ Mvc\ Model\ Resultset  
Returns the error messages produced by a batch operation

public *boolean* **delete** ([*Closure* \$conditionCallback]) inherited from Phalcon\ Mvc\ Model\ Resultset  
Deletes every record in the resultset

public *Phalcon\ Mvc\ Model* [] **filter** (*callback* \$filter) inherited from Phalcon\ Mvc\ Model\ Resultset  
Filters a resultset returning only those the developer requires

<?php

```
$filtered = $robots->filter(function($robot){  
    if ($robot->id < 3) {  
        return $robot;  
    }  
});
```

## 2.54.181 Class Phalcon\ Mvc\ Model\ Row

*implements* ArrayAccess, Countable, *Phalcon\ Mvc\ Model\ ResultInterface*

This component allows Phalcon\ Mvc\ Model to return rows without an associated entity. This objects implements the ArrayAccess interface to allow access the object as object->x or array[x].

## Methods

**public setDirtyState (int \$dirtyState)**

Set the current object's state

**public boolean offsetExists (int \$index)**

Checks whether offset exists in the row

**public string/PhalconMvcModelInterface offsetGet (int \$index)**

Gets a record in a specific position of the row

**public offsetSet (int \$index, Phalcon|Mvc|ModelInterface \$value)**

Rows cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

**public offsetUnset (int \$offset)**

Rows cannot be changed. It has only been implemented to meet the definition of the ArrayAccess interface

**public array toArray ()**

Returns the instance as an array representation

**public int count ()**

Counts how many properties were added to the row

## 2.54.182 Class Phalcon\ Mvc\ Model\ Transaction

*implements Phalcon|Mvc|Model|TransactionInterface*

Transactions are protective blocks where SQL statements are only permanent if they can all succeed as one atomic action. Phalcon\ Transaction is intended to be used with Phalcon\\_Model\\_Base. Phalcon Transactions should be created using Phalcon\ Transaction\ Manager.

```
<?php

try {

    $manager = new Phalcon\ Mvc\ Model\ Transaction\ Manager();

    $transaction = $manager->get();

    $robot = new Robots();
    $robot->setTransaction($transaction);
    $robot->name = 'WALL·E';
    $robot->created_at = date('Y-m-d');
    if ($robot->save() == false) {
        $transaction->rollback("Can't save robot");
    }

    $robotPart = new RobotParts();
    $robotPart->setTransaction($transaction);
    $robotPart->type = 'head';
    if ($robotPart->save() == false) {
        $transaction->rollback("Can't save robot part");
    }

    $transaction->commit();
}
```

```
    } catch(Phalcon\Mvc\Model\Transaction\Failed $e) {
        echo 'Failed, reason: ', $e->getMessage();
    }
}
```

## Methods

public **\_\_construct** (*Phalcon\DiInterface* \$dependencyInjector, [*boolean* \$autoBegin], [*string* \$service])

Phalcon\Mvc\Model\Transaction constructor

public **setTransactionManager** (*Phalcon\ Mvc\ Model\ Transaction\ ManagerInterface* \$manager)

Sets transaction manager related to the transaction

public *boolean* **begin** ()

Starts the transaction

public *boolean* **commit** ()

Commits the transaction

public *boolean* **rollback** ([*string* \$rollbackMessage], [*Phalcon\ Mvc\ ModelInterface* \$rollbackRecord])

Rolls back the transaction

public *Phalcon\ Db\ AdapterInterface* **getConnection** ()

Returns the connection related to transaction

public **setIsNewTransaction** (*boolean* \$isNew)

Sets if is a reused transaction or new once

public **setRollbackOnAbort** (*boolean* \$rollbackOnAbort)

Sets flag to rollback on abort the HTTP connection

public *boolean* **isManaged** ()

Checks whether transaction is managed by a transaction manager

public *array* **getMessages** ()

Returns validations messages from last save try

public *boolean* **isValid** ()

Checks whether internal connection is under an active transaction

public **setRollbackedRecord** (*Phalcon\ Mvc\ ModelInterface* \$record)

Sets object which generates rollback action

### 2.54.183 Class Phalcon\Mvc\Model\Transaction\Exception

*extends class Phalcon\ Mvc\ Model\ Exception*

Exceptions thrown in Phalcon\Mvc\Model\Transaction will use this class

## Methods

final private *Exception* **`__clone()`** inherited from *Exception*  
Clone the exception

public **`__construct ([string $message], [int $code], [Exception $previous])`** inherited from *Exception*  
*Exception* constructor

final public *string* **`getMessage()`** inherited from *Exception*  
Gets the *Exception* message

final public *int* **`getCode()`** inherited from *Exception*  
Gets the *Exception* code

final public *string* **`getFile()`** inherited from *Exception*  
Gets the file in which the exception occurred

final public *int* **`getLine()`** inherited from *Exception*  
Gets the line in which the exception occurred

final public *array* **`getTrace()`** inherited from *Exception*  
Gets the stack trace

final public *Exception* **`getPrevious()`** inherited from *Exception*  
Returns previous *Exception*

final public *Exception* **`getTraceAsString()`** inherited from *Exception*  
Gets the stack trace as a string

public *string* **`__toString()`** inherited from *Exception*  
String representation of the exception

## 2.54.184 Class Phalcon\Mvc\Model\Transaction\Failed

*extends class Phalcon\ Mvc\ Model\ Transaction\ Exception*

This class will be thrown to exit a try/catch block for isolated transactions

## Methods

public **`__construct (string $message, Phalcon\ Mvc\ ModelInterface $record)`**  
*Phalcon\ Mvc\ Model\ Transaction\ Failed* constructor

public *Phalcon\ Mvc\ Model\ MessageInterface* [] **`getRecordMessages()`**  
Returns validation record messages which stop the transaction

public *Phalcon\ Mvc\ ModelInterface* **`getRecord()`**  
Returns validation record messages which stop the transaction

final private *Exception* **`__clone()`** inherited from *Exception*  
Clone the exception

final public *string* **getMessage** () inherited from Exception  
Gets the Exception message

final public *int* **getCode** () inherited from Exception  
Gets the Exception code

final public *string* **getFile** () inherited from Exception  
Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception  
Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception  
Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception  
Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception  
Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception  
String representation of the exception

## 2.54.185 Class Phalcon\Mvc\Model\Transaction\Manager

*implements Phalcon\ Mvc\ Model\ Transaction\ ManagerInterface, Phalcon\ DI\ InjectionAwareInterface*

A transaction acts on a single database connection. If you have multiple class-specific databases, the transaction will not protect interaction among them. This class manages the objects that compose a transaction. A transaction produces a unique connection that is passed to every object part of the transaction.

```
<?php

try {

    use Phalcon\Mvc\Model\Transaction\Manager as TransactionManager;

    $transactionManager = new TransactionManager();

    $transaction = $transactionManager->get();

    $robot = new Robots();
    $robot->setTransaction($transaction);
    $robot->name = 'WALL·E';
    $robot->created_at = date('Y-m-d');
    if($robot->save()==false){
        $transaction->rollback("Can't save robot");
    }

    $robotPart = new RobotParts();
    $robotPart->setTransaction($transaction);
    $robotPart->type = 'head';
    if($robotPart->save()==false){
        $transaction->rollback("Can't save robot part");
    }
}
```

```

    }

    $transaction->commit();

}

catch(Phalcon\Mvc\Model\Transaction\Failed $e){
    echo 'Failed, reason: ', $e->getMessage();
}

```

## Methods

**public \_\_construct ([*Phalcon|DiInterface* \$dependencyInjector])**

Phalcon\Mvc\Model\Transaction\Manager constructor

**public setDI (*Phalcon|DiInterface* \$dependencyInjector)**

Sets the dependency injection container

**public *Phalcon|DiInterface* getDI ()**

Returns the dependency injection container

**public *Phalcon|Mvc|Model|Transaction|Manager* setDbService (*string* \$service)**

Sets the database service used to run the isolated transactions

**public *string* getDbService ()**

Returns the database service used to isolate the transaction

**public *Phalcon|Mvc|Model|Transaction|Manager* setRollbackPendent (*boolean* \$rollbackPendent)**

Set if the transaction manager must register a shutdown function to clean up pendent transactions

**public *boolean* getRollbackPendent ()**

Check if the transaction manager is registering a shutdown function to clean up pendent transactions

**public *boolean* has ()**

Checks whether the manager has an active transaction

**public *Phalcon|Mvc|Model|TransactionInterface* get ([*boolean* \$autoBegin])**

Returns a new Phalcon\Mvc\Model\Transaction or an already created once This method registers a shutdown function to rollback active connections

**public *Phalcon|Mvc|Model|TransactionInterface* getOrCreateTransaction ([*boolean* \$autoBegin])**

Create/Returns a new transaction or an existing one

**public *rollbackPendent ()***

Rollbacks active transactions within the manager

**public *commit ()***

Commmits active transactions within the manager

**public *rollback ([boolean \$collect])***

Rollbacks active transactions within the manager Collect will remove transaction from the manager

**public *notifyRollback (*Phalcon|Mvc|Model|TransactionInterface* \$transaction)***

Notifies the manager about a rolledback transaction

public **notifyCommit** (*Phalcon\ Mvc\ Model\ TransactionInterface* \$transaction)

Notifies the manager about a committed transaction

protected **\_collectTransaction** ()

Removes transactions from the TransactionManager

public **collectTransactions** ()

Remove all the transactions from the manager

## 2.54.186 Class Phalcon\Mvc\Model\ValidationFailed

*extends class Phalcon\ Mvc\ Model\ Exception*

This exception is generated when a model fails to save a record Phalcon\ Mvc\ Model must be set up to have this behavior

### Methods

public **\_\_construct** (*Phalcon\ Mvc\ Model* \$model, *Phalcon\ Mvc\ Model\ Message[]* \$validationMessages)

Phalcon\ Mvc\ Model\ ValidationFailed constructor

public *Phalcon\ Mvc\ Model\ Message* [] **getMessages** ()

Returns the complete group of messages produced in the validation

public *Phalcon\ Mvc\ Model* **getModel** ()

Returns the model that generated the messages

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

### 2.54.187 Abstract class Phalcon\Mvc\Model\Validator

*implements Phalcon\ Mvc\ Model\ ValidatorInterface*

This is a base class for Phalcon\Mvc\Model validators

#### Methods

**public \_\_construct (array \$options)**

Phalcon\Mvc\Model\Validator constructor

**protected appendMessage ()**

Appends a message to the validator

**public array getMessages ()**

Returns messages generated by the validator

**public array getOptions ()**

Returns all the options from the validator

**public mixed getOption ()**

Returns an option

**public boolean isSetOption ()**

Check whether a option has been defined in the validator options

**abstract public boolean validate (Phalcon\ Mvc\ ModelInterface \$record)** inherited from Phalcon\Mvc\Model\ValidatorInterface

Executes the validator

### 2.54.188 Class Phalcon\Mvc\Model\Validator\Email

*extends abstract class Phalcon\ Mvc\ Model\ Validator*

*implements Phalcon\ Mvc\ Model\ ValidatorInterface*

Allows to validate if email fields has correct values

<?php

```
use Phalcon\Mvc\Model\Validator\Email as EmailValidator;

class Subscriptors extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new EmailValidator(array(
            'field' => 'electronic_mail',
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

```
        }
    }
}
```

## Methods

**public boolean validate (*Phalcon|Mvc|ModelInterface* \$record)**

Executes the validator

**public \_\_construct (*array* \$options)** inherited from Phalcon\Mvc\Model\Validator

Phalcon\Mvc\Model\Validator constructor

**protected appendMessage ()** inherited from Phalcon\Mvc\Model\Validator

Appends a message to the validator

**public array getMessages ()** inherited from Phalcon\Mvc\Model\Validator

Returns messages generated by the validator

**public array getOptions ()** inherited from Phalcon\Mvc\Model\Validator

Returns all the options from the validator

**public mixed getOption ()** inherited from Phalcon\Mvc\Model\Validator

Returns an option

**public boolean isSetOption ()** inherited from Phalcon\Mvc\Model\Validator

Check whether a option has been defined in the validator options

## 2.54.189 Class Phalcon\Mvc\Model\Validator\Exclusionin

*extends abstract class Phalcon\ Mvc\ Model\ Validator*

*implements Phalcon\ Mvc\ Model\ ValidatorInterface*

Phalcon\Mvc\Model\Validator\ExclusionIn Check if a value is not included into a list of values

<?php

```
use Phalcon\Mvc\Model\Validator\ExclusionIn as ExclusionInValidator;
```

```
class Subscriptors extends Phalcon\Mvc\Model
{
```

```
    public function validation()
    {
        $this->validate(new ExclusionInValidator(array(
            'field' => 'status',
            'domain' => array('A', 'I')
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

}

## Methods

`public boolean validate (Phalcon|Mvc|ModelInterface $record)`

Executes the validator

`public __construct (array $options) inherited from Phalcon\ Mvc\ Model\ Validator`

Phalcon\ Mvc\ Model\ Validator constructor

`protected appendMessage () inherited from Phalcon\ Mvc\ Model\ Validator`

Appends a message to the validator

`public array getMessages () inherited from Phalcon\ Mvc\ Model\ Validator`

Returns messages generated by the validator

`public array getOptions () inherited from Phalcon\ Mvc\ Model\ Validator`

Returns all the options from the validator

`public mixed getOption () inherited from Phalcon\ Mvc\ Model\ Validator`

Returns an option

`public boolean isSetOption () inherited from Phalcon\ Mvc\ Model\ Validator`

Check whether a option has been defined in the validator options

## 2.54.190 Class Phalcon\ Mvc\ Model\ Validator\ InclusionIn

`extends abstract class Phalcon\ Mvc\ Model\ Validator`

`implements Phalcon\ Mvc\ Model\ ValidatorInterface`

Phalcon\ Mvc\ Model\ Validator\ InclusionIn Check if a value is included into a list of values

`<?php`

```
use Phalcon\ Mvc\ Model\ Validator\ InclusionIn as InclusionInValidator;
```

```
class Subscriptors extends Phalcon\ Mvc\ Model
{
```

```
    public function validation()
    {
        $this->validate(new InclusionInValidator(array(
            'field' => 'status',
            'domain' => array('A', 'I')
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

}

## Methods

`public boolean validate (Phalcon|Mvc|ModelInterface $record)`

Executes validator

`public __construct (array $options) inherited from Phalcon\ Mvc\ Model\ Validator`

Phalcon\ Mvc\ Model\ Validator constructor

`protected appendMessage () inherited from Phalcon\ Mvc\ Model\ Validator`

Appends a message to the validator

`public array getMessages () inherited from Phalcon\ Mvc\ Model\ Validator`

Returns messages generated by the validator

`public array getOptions () inherited from Phalcon\ Mvc\ Model\ Validator`

Returns all the options from the validator

`public mixed getOption () inherited from Phalcon\ Mvc\ Model\ Validator`

Returns an option

`public boolean isSetOption () inherited from Phalcon\ Mvc\ Model\ Validator`

Check whether a option has been defined in the validator options

## 2.54.191 Class Phalcon\ Mvc\ Model\ Validator\ Numericality

*extends abstract class Phalcon\ Mvc\ Model\ Validator*

*implements Phalcon\ Mvc\ Model\ ValidatorInterface*

Allows to validate if a field has a valid numeric format

<?php

```
use Phalcon\ Mvc\ Model\ Validator\ Numericality as NumericalityValidator;

class Products extends Phalcon\ Mvc\ Model
{

    public function validation()
    {
        $this->validate(new NumericalityValidator(array(
            'field' => 'price'
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

## Methods

`public boolean validate (Phalcon|Mvc|ModelInterface $record)`

Executes the validator

public **\_\_construct** (*array \$options*) inherited from Phalcon\Mvc\Model\Validator  
 Phalcon\Mvc\Model\Validator constructor

protected **appendMessage** () inherited from Phalcon\Mvc\Model\Validator  
 Appends a message to the validator

public *array getMessages* () inherited from Phalcon\Mvc\Model\Validator  
 Returns messages generated by the validator

public *array getOptions* () inherited from Phalcon\Mvc\Model\Validator  
 Returns all the options from the validator

public *mixed getOption* () inherited from Phalcon\Mvc\Model\Validator  
 Returns an option

public *boolean isSetOption* () inherited from Phalcon\Mvc\Model\Validator  
 Check whether a option has been defined in the validator options

## 2.54.192 Class Phalcon\Mvc\Model\Validator\PresenceOf

*extends abstract class Phalcon\ Mvc\ Model\ Validator*  
*implements Phalcon\ Mvc\ Model\ ValidatorInterface*

Allows to validate if a filed have a value different of null and empty string ("")

<?php

```
use Phalcon\Mvc\Model\Validator\PresenceOf;

class Subscribers extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new PresenceOf(array(
            'field' => 'name',
            'message' => 'The name is required'
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

### Methods

public *boolean validate* (*Phalcon\ Mvc\ ModelInterface \$record*)

Executes the validator

public **\_\_construct** (*array \$options*) inherited from Phalcon\Mvc\Model\Validator

Phalcon\Mvc\Model\Validator constructor

protected **appendMessage** () inherited from Phalcon\Mvc\Model\Validator

Appends a message to the validator

public *array* **getMessages** () inherited from Phalcon\Mvc\Model\Validator

Returns messages generated by the validator

public *array* **getOptions** () inherited from Phalcon\Mvc\Model\Validator

Returns all the options from the validator

public *mixed* **getOption** () inherited from Phalcon\Mvc\Model\Validator

Returns an option

public *boolean* **isSetOption** () inherited from Phalcon\Mvc\Model\Validator

Check whether a option has been defined in the validator options

## 2.54.193 Class Phalcon\Mvc\Model\Validator\Regex

*extends* abstract class Phalcon\ Mvc\ Model\ Validator

*implements* Phalcon\ Mvc\ Model\ ValidatorInterface

Allows validate if the value of a field matches a regular expression

<?php

```
use Phalcon\Mvc\Model\Validator\Regex as RegexValidator;

class Subscribers extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new RegexValidator(array(
            'field' => 'created_at',
            'pattern' => '/^([0-9]{4})[-/]([01-9]|1[12])[-/]([01-9]|1[12]) [0-9]{2} [0-9]{2}/,
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

### Methods

public *boolean* **validate** (*Phalcon\ Mvc\ ModelInterface* \$record)

Executes the validator

public **\_\_construct** (*array* \$options) inherited from Phalcon\Mvc\Model\Validator

Phalcon\Mvc\Model\Validator constructor

protected **appendMessage** () inherited from Phalcon\Mvc\Model\Validator

Appends a message to the validator

public *array* **getMessages** () inherited from Phalcon\Mvc\Model\Validator

Returns messages generated by the validator

`public array getOptions ()` inherited from Phalcon\Mvc\Model\Validator

Returns all the options from the validator

`public mixed getOption ()` inherited from Phalcon\Mvc\Model\Validator

Returns an option

`public boolean isSetOption ()` inherited from Phalcon\Mvc\Model\Validator

Check whether a option has been defined in the validator options

## 2.54.194 Class Phalcon\Mvc\Model\Validator\StringLength

*extends abstract class Phalcon\ Mvc\ Model\ Validator*

*implements Phalcon\ Mvc\ Model\ ValidatorInterface*

Simply validates specified string length constraints

<?php

```
use Phalcon\Mvc\Model\Validator\StringLength as StringLengthValidator;

class Subscriptors extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new StringLengthValidator(array(
            'field' => 'name_last',
            'max' => 50,
            'min' => 2,
            'messageMaximum' => 'We don\'t like really long names',
            'messageMinimum' => 'We want more than just their initials',
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

### Methods

`public boolean validate (Phalcon\ Mvc\ ModelInterface $record)`

Executes the validator

`public __construct (array $options)` inherited from Phalcon\Mvc\Model\Validator

Phalcon\Mvc\Model\Validator constructor

`protected appendMessage ()` inherited from Phalcon\Mvc\Model\Validator

Appends a message to the validator

`public array getMessages ()` inherited from Phalcon\Mvc\Model\Validator

Returns messages generated by the validator

`public array getOptions ()` inherited from Phalcon\Mvc\Model\Validator

Returns all the options from the validator

public *mixed* **getOption** () inherited from Phalcon\Mvc\Model\Validator

Returns an option

public *boolean* **isSetOption** () inherited from Phalcon\Mvc\Model\Validator

Check whether a option has been defined in the validator options

## 2.54.195 Class Phalcon\Mvc\Model\Validator\Uniqueness

*extends* abstract class Phalcon\Mvc\Model\Validator

*implements* Phalcon\Mvc\Model\ValidatorInterface

Validates that a field or a combination of a set of fields are not present more than once in the existing records of the related table

```
<?php

use Phalcon\Mvc\Model\Validator\Uniqueness as Uniqueness;

class Subscribers extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new Uniqueness(array(
            'field' => 'email',
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

### Methods

public *boolean* **validate** (*Phalcon\Mvc\ModelInterface* \$record)

Executes the validator

public **\_\_construct** (*array* \$options) inherited from Phalcon\ Mvc\ Model\ Validator

Phalcon\ Mvc\ Model\ Validator constructor

protected **appendMessage** () inherited from Phalcon\ Mvc\ Model\ Validator

Appends a message to the validator

public *array* **getMessages** () inherited from Phalcon\ Mvc\ Model\ Validator

Returns messages generated by the validator

public *array* **getOptions** () inherited from Phalcon\ Mvc\ Model\ Validator

Returns all the options from the validator

public *mixed* **getOption** () inherited from Phalcon\ Mvc\ Model\ Validator

Returns an option

`public boolean isSetOption ()` inherited from Phalcon\Mvc\Model\Validator

Check whether a option has been defined in the validator options

## 2.54.196 Class Phalcon\Mvc\Model\Validator\Url

*extends abstract class Phalcon\Mvc\Model\Validator*

*implements Phalcon\Mvc\Model\ValidatorInterface*

Allows to validate if a field has a url format

<?php

```
use Phalcon\Mvc\Model\Validator\Url as UrlValidator;

class Posts extends Phalcon\Mvc\Model
{

    public function validation()
    {
        $this->validate(new UrlValidator(array(
            'field' => 'source_url',
        )));
        if ($this->validationHasFailed() == true) {
            return false;
        }
    }
}
```

### Methods

`public boolean validate (Phalcon\Mvc\ModelInterface $record)`

Executes the validator

`public __construct (array $options)` inherited from Phalcon\Mvc\Model\Validator

Phalcon\Mvc\Model\Validator constructor

`protected appendMessage ()` inherited from Phalcon\Mvc\Model\Validator

Appends a message to the validator

`public array getMessages ()` inherited from Phalcon\Mvc\Model\Validator

Returns messages generated by the validator

`public array getOptions ()` inherited from Phalcon\Mvc\Model\Validator

Returns all the options from the validator

`public mixed getOption ()` inherited from Phalcon\Mvc\Model\Validator

Returns an option

`public boolean isSetOption ()` inherited from Phalcon\Mvc\Model\Validator

Check whether a option has been defined in the validator options

## 2.54.197 Class Phalcon\Mvc\Router

*implements Phalcon\ Mvc\ RouterInterface, Phalcon\ DI\ InjectionAwareInterface*

Phalcon\Mvc\Router is the standard framework router. Routing is the process of taking a URI endpoint (that part of the URI which comes after the base URL) and decomposing it into parameters to determine which module, controller, and action of that controller should receive the request

<?php

```
$router = new Phalcon\Mvc\Router();

$router->add(
    "/documentation/{chapter}/{name}.{type:[a-z]+}",
    array(
        "controller" => "documentation",
        "action"      => "show"
    )
);

$router->handle();

echo $router->getControllerName();
```

### Constants

*integer URI\_SOURCE\_GET\_URL*

*integer URI\_SOURCE\_SERVER\_REQUEST\_URI*

### Methods

**public \_\_construct ([boolean \$defaultRoutes])**

Phalcon\Mvc\Router constructor

**public setDI (Phalcon\DiInterface \$dependencyInjector)**

Sets the dependency injector

**public Phalcon\DiInterface getDI ()**

Returns the internal dependency injector

**public string getRewriteUri ()**

Get rewrite info. This info is read from \$\_GET['\_url']. This returns '/' if the rewrite information cannot be read

**public Phalcon\ Mvc\ Router setUriSource (int \$uriSource)**

Sets the URI source. One of the URI\_SOURCE\_\* constants

<?php

```
$router->setUriSource(Router::URI_SOURCE_SERVER_REQUEST_URI);
```

**public Phalcon\ Mvc\ Router removeExtraSlashes (boolean \$remove)**

Set whether router must remove the extra slashes in the handled routes

public *Phalcon\ Mvc\ Router* **setDefaultNamespace** (*string* \$namespaceName)  
Sets the name of the default namespace

public *string* **getDefaultNamespace** ()  
Returns the name of the default namespace

public *Phalcon\ Mvc\ Router* **setDefaultModule** (*string* \$moduleName)  
Sets the name of the default module

public *string* **getDefaultModule** ()  
Returns the name of the default module

public *Phalcon\ Mvc\ Router* **setDefaultController** (*string* \$controllerName)  
Sets the default controller name

public *string* **getDefaultController** ()  
Returns the default controller name

public *Phalcon\ Mvc\ Router* **setDefaultAction** (*string* \$actionName)  
Sets the default action name

public *string* **getDefaultAction** ()  
Returns the default action name

public *Phalcon\ Mvc\ Router* **setDefaults** (*array* \$defaults)  
Sets an array of default paths. If a route is missing a path the router will use the defined here This method must not be used to set a 404 route

&lt;?php

```
$router->setDefaults(array(
    'module' => 'common',
    'action' => 'index'
));
```

public *array* **getDefaults** ()

Returns an array of default parameters

public **handle** ([*string* \$uri])

Handles routing information received from the rewrite engine

&lt;?php

```
//Read the info from the rewrite engine
$router->handle();
```

//Manually passing an URL

```
$router->handle('/posts/edit/1');
```

public *Phalcon\ Mvc\ Router\ Route* **add** (*string* \$pattern, [*string/array* \$paths], [*string* \$httpMethods])

Adds a route to the router without any HTTP constraint

&lt;?php

```
$router->add('/about', 'About::index');
```

public *Phalcon\ Mvc\ Router\ Route* **addGet** (*string \$pattern, [string/array \$paths]*)  
Adds a route to the router that only match if the HTTP method is GET

public *Phalcon\ Mvc\ Router\ Route* **addPost** (*string \$pattern, [string/array \$paths]*)  
Adds a route to the router that only match if the HTTP method is POST

public *Phalcon\ Mvc\ Router\ Route* **addPut** (*string \$pattern, [string/array \$paths]*)  
Adds a route to the router that only match if the HTTP method is PUT

public *Phalcon\ Mvc\ Router\ Route* **addPatch** (*string \$pattern, [string/array \$paths]*)  
Adds a route to the router that only match if the HTTP method is PATCH

public *Phalcon\ Mvc\ Router\ Route* **addDelete** (*string \$pattern, [string/array \$paths]*)  
Adds a route to the router that only match if the HTTP method is DELETE

public *Phalcon\ Mvc\ Router\ Route* **addOptions** (*string \$pattern, [string/array \$paths]*)  
Add a route to the router that only match if the HTTP method is OPTIONS

public *Phalcon\ Mvc\ Router\ Route* **addHead** (*string \$pattern, [string/array \$paths]*)  
Adds a route to the router that only match if the HTTP method is HEAD

public *Phalcon\ Mvc\ Router* **mount** (*unknown \$group*)  
Mounts a group of routes in the router

public *Phalcon\ Mvc\ Router* **notFound** (*array/string \$paths*)  
Set a group of paths to be returned when none of the defined routes are matched

public **clear** ()  
Removes all the pre-defined routes

public *string* **getNamespaceName** ()  
Returns the processed namespace name

public *string* **getModuleName** ()  
Returns the processed module name

public *string* **getControllerName** ()  
Returns the processed controller name

public *string* **getActionName** ()  
Returns the processed action name

public *array* **getParams** ()  
Returns the processed parameters

public *Phalcon\ Mvc\ Router\ Route* **getMatchedRoute** ()  
Returns the route that matchs the handled URI

public *array* **getMatches** ()  
Returns the sub expressions in the regular expression matched

public *bool* **wasMatched** ()  
Checks if the router macthes any of the defined routes

```
public Phalcon\Mvc\Router\Route [] getRoutes ()
>Returns all the routes defined in the router

public Phalcon\Mvc\Router\Route | false getRouteById (string $id)
>Returns a route object by its id

public Phalcon\Mvc\Router\Route getRouteByName (string $name)
>Returns a route object by its name

public isExactControllerName ()
>Returns whether controller name should not be mangled
```

## 2.54.198 Class Phalcon\Mvc\Router\Annotations

*extends class Phalcon\Mvc\Router  
implements Phalcon\DI\InjectionAwareInterface, Phalcon\Mvc\RouterInterface*

A router that reads routes annotations from classes/resources

```
<?php

$di['router'] = function() {

    //Use the annotations router
    $router = new \Phalcon\Mvc\Router\Annotations(false);

    //This will do the same as above but only if the handled uri starts with /robots
    $router->addResource('Robots', '/robots');

    return $router;
};
```

### Constants

```
integer URI_SOURCE_GET_URL
integer URI_SOURCE_SERVER_REQUEST_URI
```

### Methods

```
public Phalcon\Mvc\Router\Annotations addResource (string $handler, [string $prefix])
```

Adds a resource to the annotations handler A resource is a class that contains routing annotations

```
public Phalcon\Mvc\Router\Annotations addModuleResource (string $module, string $handler, [string $prefix])
```

Adds a resource to the annotations handler A resource is a class that contains routing annotations The class is located in a module

```
public handle ([string $uri])
```

Produce the routing parameters from the rewrite information

```
public processControllerAnnotation (string $handler, unknown $annotation)
```

Checks for annotations in the controller docblock

```
public processActionAnnotation (string $module, string $namespace, string $controller, string $action,  
Phalcon\Annotations\Annotation $annotation)
```

Checks for annotations in the public methods of the controller

```
public setControllerSuffix (string $controllerSuffix)
```

Changes the controller class suffix

```
public setActionSuffix (string $actionSuffix)
```

Changes the action method suffix

```
public array getResources ()
```

Return the registered resources

```
public __construct ([boolean $defaultRoutes]) inherited from Phalcon\Mvc\Router
```

Phalcon\Mvc\Router constructor

```
public setDI (Phalcon\DiInterface $dependencyInjector) inherited from Phalcon\Mvc\Router
```

Sets the dependency injector

```
public Phalcon\DiInterface getDI () inherited from Phalcon\Mvc\Router
```

Returns the internal dependency injector

```
public string getRewriteUri () inherited from Phalcon\Mvc\Router
```

Get rewrite info. This info is read from \$\_GET['url']. This returns '/' if the rewrite information cannot be read

```
public Phalcon\Mvc\Router setUriSource (int $uriSource) inherited from Phalcon\Mvc\Router
```

Sets the URI source. One of the URI\_SOURCE\_\* constants

```
<?php
```

```
$router->setUriSource(Router::URI_SOURCE_SERVER_REQUEST_URI);
```

```
public Phalcon\Mvc\Router removeExtraSlashes (boolean $remove) inherited from Phalcon\Mvc\Router
```

Set whether router must remove the extra slashes in the handled routes

```
public Phalcon\Mvc\Router setDefaultNamespace (string $namespaceName) inherited from  
Phalcon\Mvc\Router
```

Sets the name of the default namespace

```
public string getDefaultNamespace () inherited from Phalcon\Mvc\Router
```

Returns the name of the default namespace

```
public Phalcon\Mvc\Router setDefaultModule (string $moduleName) inherited from  
Phalcon\Mvc\Router
```

Sets the name of the default module

```
public string getDefaultModule () inherited from Phalcon\Mvc\Router
```

Returns the name of the default module

```
public Phalcon\Mvc\Router setDefaultController (string $controllerName) inherited from  
Phalcon\Mvc\Router
```

Sets the default controller name

`public string getDefaultController ()` inherited from Phalcon\Mvc\Router

Returns the default controller name

`public Phalcon\ Mvc\ Router setDefaultAction (string $actionName)` inherited from Phalcon\Mvc\Router

Sets the default action name

`public string getDefaultAction ()` inherited from Phalcon\Mvc\Router

Returns the default action name

`public Phalcon\ Mvc\ Router setDefaults (array $defaults)` inherited from Phalcon\Mvc\Router

Sets an array of default paths. If a route is missing a path the router will use the defined here This method must not be used to set a 404 route

<?php

```
$router->setDefaults(array(
    'module' => 'common',
    'action' => 'index'
));
```

`public array getDefaults ()` inherited from Phalcon\Mvc\Router

Returns an array of default parameters

`public Phalcon\ Mvc\ Router\ Route add (string $pattern, [string/array $paths], [string $httpMethods])` inherited from Phalcon\Mvc\Router

Adds a route to the router without any HTTP constraint

<?php

```
$router->add('/about', 'About::index');
```

`public Phalcon\ Mvc\ Router\ Route addGet (string $pattern, [string/array $paths])` inherited from Phalcon\Mvc\Router

Adds a route to the router that only match if the HTTP method is GET

`public Phalcon\ Mvc\ Router\ Route addPost (string $pattern, [string/array $paths])` inherited from Phalcon\Mvc\Router

Adds a route to the router that only match if the HTTP method is POST

`public Phalcon\ Mvc\ Router\ Route addPut (string $pattern, [string/array $paths])` inherited from Phalcon\Mvc\Router

Adds a route to the router that only match if the HTTP method is PUT

`public Phalcon\ Mvc\ Router\ Route addPatch (string $pattern, [string/array $paths])` inherited from Phalcon\Mvc\Router

Adds a route to the router that only match if the HTTP method is PATCH

`public Phalcon\ Mvc\ Router\ Route addDelete (string $pattern, [string/array $paths])` inherited from Phalcon\Mvc\Router

Adds a route to the router that only match if the HTTP method is DELETE

`public Phalcon\ Mvc\ Router\ Route addOptions (string $pattern, [string/array $paths])` inherited from Phalcon\Mvc\Router

Add a route to the router that only match if the HTTP method is OPTIONS

public *Phalcon\ Mvc\ Router\ Route* **addHead** (*string \$pattern, [string/array \$paths]*) inherited from Phalcon\ Mvc\ Router

Adds a route to the router that only match if the HTTP method is HEAD

public *Phalcon\ Mvc\ Router* **mount** (*unknown \$group*) inherited from Phalcon\ Mvc\ Router

Mounts a group of routes in the router

public *Phalcon\ Mvc\ Router* **notFound** (*array/string \$paths*) inherited from Phalcon\ Mvc\ Router

Set a group of paths to be returned when none of the defined routes are matched

public **clear** () inherited from Phalcon\ Mvc\ Router

Removes all the pre-defined routes

public *string getNamespaceName* () inherited from Phalcon\ Mvc\ Router

Returns the processed namespace name

public *string getModuleName* () inherited from Phalcon\ Mvc\ Router

Returns the processed module name

public *string getControllerName* () inherited from Phalcon\ Mvc\ Router

Returns the processed controller name

public *string getActionName* () inherited from Phalcon\ Mvc\ Router

Returns the processed action name

public *array getParams* () inherited from Phalcon\ Mvc\ Router

Returns the processed parameters

public *Phalcon\ Mvc\ Router\ Route getMatchedRoute* () inherited from Phalcon\ Mvc\ Router

Returns the route that matchs the handled URI

public *array getMatches* () inherited from Phalcon\ Mvc\ Router

Returns the sub expressions in the regular expression matched

public *bool wasMatched* () inherited from Phalcon\ Mvc\ Router

Checks if the router macthes any of the defined routes

public *Phalcon\ Mvc\ Router\ Route [] getRoutes* () inherited from Phalcon\ Mvc\ Router

Returns all the routes defined in the router

public *Phalcon\ Mvc\ Router\ Route | false getRouteById* (*string \$id*) inherited from Phalcon\ Mvc\ Router

Returns a route object by its id

public *Phalcon\ Mvc\ Router\ Route getRouteByName* (*string \$name*) inherited from Phalcon\ Mvc\ Router

Returns a route object by its name

public **isExactControllerName** () inherited from Phalcon\ Mvc\ Router

Returns whether controller name should not be mangled

## 2.54.199 Class Phalcon\Mvc\Router\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\ Mvc\ Router will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.200 Class Phalcon\Mvc\Router\Group

Helper class to create a group of routes with common attributes

```
<?php

$router = new Phalcon\Mvc\Router();

//Create a group with a common module and controller
$blog = new Phalcon\Mvc\Router\Group(array(
    'module' => 'blog',
    'controller' => 'index'
));

//All the routes start with /blog
```

```
$blog->setPrefix('/blog');

//Add a route to the group
$blog->add('/save', array(
    'action' => 'save'
));

//Add another route to the group
$blog->add('/edit/{id}', array(
    'action' => 'edit'
));

//This route maps to a controller different than the default
$blog->add('/blog', array(
    'controller' => 'about',
    'action' => 'index'
));

//Add the group to the router
$router->mount($blog);
```

## Methods

public **\_\_construct** ([array \$paths])

Phalcon\Mvc\Router\Group constructor

public *Phalcon\ Mvc\ Router\ Group* **setHostname** (string \$hostname)

Set a hostname restriction for all the routes in the group

public *string* **getHostname** ()

Returns the hostname restriction

public *Phalcon\ Mvc\ Router\ Group* **setPrefix** (string \$prefix)

Set a common uri prefix for all the routes in this group

public *string* **getPrefix** ()

Returns the common prefix for all the routes

public *Phalcon\ Mvc\ Router\ Group* **beforeMatch** (*unknown* \$beforeMatch)

Set a before-match condition for the whole group

public *string* **getBeforeMatch** ()

Returns the before-match condition if any

public *Phalcon\ Mvc\ Router\ Group* **setPaths** (array \$paths)

Set common paths for all the routes in the group

public *array/string* **getPaths** ()

Returns the common paths defined for this group

public *Phalcon\ Mvc\ Router\ Route* [] **getRoutes** ()

Returns the routes added to the group

protected *Phalcon\ Mvc\ Router\ Route* **\_addRoute** ()

Adds a route applying the common attributes

`public Phalcon\ Mvc\ Router\ Route add (string $pattern, [string/array $paths], [string $httpMethods])`

Adds a route to the router on any HTTP method

`<?php`

```
$router->add('/about', 'About::index');
```

`public Phalcon\ Mvc\ Router\ Route addGet (string $pattern, [string/array $paths])`

Adds a route to the router that only match if the HTTP method is GET

`public Phalcon\ Mvc\ Router\ Route addPost (string $pattern, [string/array $paths])`

Adds a route to the router that only match if the HTTP method is POST

`public Phalcon\ Mvc\ Router\ Route addPut (string $pattern, [string/array $paths])`

Adds a route to the router that only match if the HTTP method is PUT

`public Phalcon\ Mvc\ Router\ Route addPatch (string $pattern, [string/array $paths])`

Adds a route to the router that only match if the HTTP method is PATCH

`public Phalcon\ Mvc\ Router\ Route addDelete (string $pattern, [string/array $paths])`

Adds a route to the router that only match if the HTTP method is DELETE

`public Phalcon\ Mvc\ Router\ Route addOptions (string $pattern, [string/array $paths])`

Add a route to the router that only match if the HTTP method is OPTIONS

`public Phalcon\ Mvc\ Router\ Route addHead (string $pattern, [string/array $paths])`

Adds a route to the router that only match if the HTTP method is HEAD

`public clear ()`

Removes all the pre-defined routes

`public Phalcon\ Mvc\ Router\ Group convert (string $name, callable $converter)`

Adds a converter to perform an additional transformation for certain parameter

`public array/null getConverters ()`

Returns the router converter

`public Phalcon\ Mvc\ Router\ Group setName (unknown $name)`

Set the name of the group

`public string getName ()`

Returns the name of this group

## 2.54.201 Class Phalcon\ Mvc\ Router\ Route

*implements Phalcon\ Mvc\ Router\ RouteInterface*

This class represents every route added to the router

## Methods

public **\_\_construct** (*string* \$pattern, [*array* \$paths], [*array/string* \$httpMethods])

Phalcon\Mvc\Router\Route constructor

public *string* **compilePattern** (*string* \$pattern)

Replaces placeholders from pattern returning a valid PCRE regular expression

public *Phalcon|Mvc|Router|Route* **via** (*string/array* \$httpMethods)

Set one or more HTTP methods that constraint the matching of the route

<?php

```
$route->via('GET');  
$route->via(array('GET', 'POST'));
```

public **reConfigure** (*string* \$pattern, [*array* \$paths])

Reconfigure the route adding a new pattern and a set of paths

public *string* **getName** ()

Returns the route's name

public *Phalcon|Mvc|Router|Route* **setName** (*string* \$name)

Sets the route's name

<?php

```
$router->add('/about', array(  
    'controller' => 'about',  
)>>setName('about');
```

public *Phalcon|Mvc|Router|Route* **beforeMatch** (*callback* \$callback)

Sets a callback that is called if the route is matched. The developer can implement any arbitrary conditions here If the callback returns false the route is treated as not matched

public *mixed* **getBeforeMatch** ()

Returns the 'before match' callback if any

public *string* **getRouteId** ()

Returns the route's id

public *string* **getPattern** ()

Returns the route's pattern

public *string* **getCompiledPattern** ()

Returns the route's compiled pattern

public *array* **getPaths** ()

Returns the paths

public *array* **getReversedPaths** ()

Returns the paths using positions as keys and names as values

public *Phalcon|Mvc|Router|Route* **setHttpMethods** (*string/array* \$httpMethods)

Sets a set of HTTP methods that constraint the matching of the route (alias of via)

```
<?php
$route->setHttpMethods('GET');
$route->setHttpMethods(array('GET', 'POST'));
```

**public string/array getHttpMethods ()**

Returns the HTTP methods that constraint matching the route

**public Phalcon\ Mvc\ Router\ Route setHostname (unknown \$hostname)**

Sets a hostname restriction to the route

```
<?php
```

```
$route->setHostname('localhost');
```

**public string getHostname ()**

Returns the hostname restriction if any

**public Phalcon\ Mvc\ RouteInterface setGroup (Phalcon\ Mvc\ Router\ Group \$group)**

Sets the group associated with the route

**public Phalcon\ Mvc\ Router\ Group |null getGroup ()**

Returns the group associated with the route

**public Phalcon\ Mvc\ Router\ Route convert (string \$name, callable \$converter)**

Adds a converter to perform an additional transformation for certain parameter

**public array getConverters ()**

Returns the router converter

**public static reset ()**

Resets the internal route id generator

## 2.54.202 Class Phalcon\ Mvc\ Url

*implements Phalcon\ Mvc\ UrlInterface, Phalcon\ DI\ InjectionAwareInterface*

This components aids in the generation of: URIs, URLs and Paths

```
<?php
```

```
//Generate a URL appending the URI to the base URI
echo $url->get('products/edit/1');

//Generate a URL for a predefined route
echo $url->get(array('for' => 'blog-post', 'title' => 'some-cool-stuff', 'year' => '2012'));
```

### Methods

**public setDI (Phalcon\ DiInterface \$dependencyInjector)**

Sets the DependencyInjector container

public *Phalcon\DiInterface* **getDI** ()

Returns the DependencyInjector container

public *Phalcon\Mvc\Url* **setBaseUri** (*string* \$baseUri)

Sets a prefix for all the URIs to be generated

<?php

```
$url->setBaseUri('/invo/');
```

```
$url->setBaseUri('/invo/index.php');
```

public *Phalcon\Mvc\Url* **setStaticBaseUri** (*string* \$staticBaseUri)

Sets a prefix for all static URLs generated

<?php

```
$url->setStaticBaseUri('/invo/');
```

public *string* **getBaseUri** ()

Returns the prefix for all the generated urls. By default /

public *string* **getStaticBaseUri** ()

Returns the prefix for all the generated static urls. By default /

public *Phalcon\Mvc\Url* **setbasePath** (*string* \$basePath)

Sets a base path for all the generated paths

<?php

```
$url->setbasePath('/var/www/htdocs/');
```

public *string* **getbasePath** ()

Returns the base path

public *string* **get** ([*string/array* \$uri], [*unknown* \$args], [*bool/null* \$local])

Generates a URL

<?php

```
//Generate a URL appending the URI to the base URI
echo $url->get('products/edit/1');
```

```
//Generate a URL for a predefined route
echo $url->get(array('for' => 'blog-post', 'title' => 'some-cool-stuff', 'year' => '2012'));
```

public *string* **getStatic** ([*string/array* \$uri])

Generates a URL for a static resource

public *string* **path** ([*string* \$path])

Generates a local path

## 2.54.203 Class Phalcon\Mvc\Url\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Mvc\Url will use this class

## Methods

`final private Exception __clone ()` inherited from `Exception`

Clone the exception

`public __construct ([string $message], [int $code], [Exception $previous])` inherited from `Exception`

Exception constructor

`final public string getMessage ()` inherited from `Exception`

Gets the Exception message

`final public int getCode ()` inherited from `Exception`

Gets the Exception code

`final public string getFile ()` inherited from `Exception`

Gets the file in which the exception occurred

`final public int getLine ()` inherited from `Exception`

Gets the line in which the exception occurred

`final public array getTrace ()` inherited from `Exception`

Gets the stack trace

`final public Exception getPrevious ()` inherited from `Exception`

Returns previous Exception

`final public Exception getTraceAsString ()` inherited from `Exception`

Gets the stack trace as a string

`public string __toString ()` inherited from `Exception`

String representation of the exception

## 2.54.204 Class Phalcon\Mvc\User\Component

`extends abstract class Phalcon\DI\Injectable`

`implements Phalcon\Events\EventsAwareInterface, Phalcon\DI\InjectionAwareInterface`

This class can be used to provide user components easy access to services in the application

## Methods

`public setDI (Phalcon\DiInterface $dependencyInjector)` inherited from `Phalcon\DI\Injectable`

Sets the dependency injector

`public Phalcon\DiInterface getDI ()` inherited from `Phalcon\DI\Injectable`

Returns the internal dependency injector

`public setEventsManager (Phalcon\Events\ManagerInterface $eventsManager)` inherited from `Phalcon\DI\Injectable`

Sets the event manager

public *Phalcon\\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\DI\Injectable

Returns the internal event manager

public **\_\_get** (*unknown* \$property) inherited from Phalcon\DI\Injectable

Magic method **\_\_get**

## 2.54.205 Class Phalcon\Mvc\User\Module

*extends* abstract class *Phalcon\DI\Injectable*

*implements* *Phalcon\\Events\EventsAwareInterface*, *Phalcon\DI\InjectionAwareInterface*

This class can be used to provide user modules easy access to services in the application

### Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\DI\Injectable

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\DI\Injectable

Sets the event manager

public *Phalcon\\Events\ManagerInterface* **getEventsManager** () inherited from Phalcon\DI\Injectable

Returns the internal event manager

public **\_\_get** (*unknown* \$property) inherited from Phalcon\DI\Injectable

Magic method **\_\_get**

## 2.54.206 Class Phalcon\Mvc\User\Plugin

*extends* abstract class *Phalcon\DI\Injectable*

*implements* *Phalcon\\Events\EventsAwareInterface*, *Phalcon\DI\InjectionAwareInterface*

This class can be used to provide user plugins an easy access to services in the application

### Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\DI\Injectable

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\DI\Injectable

Sets the event manager

`public Phalcon\Events\ManagerInterface getEventsManager ()` inherited from `Phalcon\DI\Injectable`

Returns the internal event manager

`public __get (unknown $property)` inherited from `Phalcon\DI\Injectable`

Magic method `__get`

## 2.54.207 Class `Phalcon\Mvc\View`

*extends abstract class `Phalcon\DI\Injectable`*

*implements `Phalcon\Events\EventsAwareInterface`, `Phalcon\DI\InjectionAwareInterface`, `Phalcon\Mvc\ViewInterface`*

`Phalcon\Mvc\View` is a class for working with the “view” portion of the model-view-controller pattern. That is, it exists to help keep the view script separate from the model and controller scripts. It provides a system of helpers, output filters, and variable escaping.

`<?php`

```
//Setting views directory
$view = new Phalcon\Mvc\View();
$view->setViewsDir('app/views/');

$view->start();
//Shows recent posts view (app/views/posts/recent.phtml)
$view->render('posts', 'recent');
$view->finish();

//Printing views output
echo $view->getContent();
```

### Constants

*integer `LEVEL_MAIN_LAYOUT`*

*integer `LEVEL_AFTER_TEMPLATE`*

*integer `LEVEL_LAYOUT`*

*integer `LEVEL_BEFORE_TEMPLATE`*

*integer `LEVEL_ACTION_VIEW`*

*integer `LEVEL_NO_RENDER`*

### Methods

`public __construct ([array $options])`

`Phalcon\Mvc\View` constructor

`public Phalcon\Mvc\View setViewsDir (string $viewsDir)`

Sets views directory. Depending of your platform, always add a trailing slash or backslash

`public string getViewsDir ()`

Gets views directory

```
public Phalcon\Mvc\View setLayoutsDir (string $layoutsDir)
```

Sets the layouts sub-directory. Must be a directory under the views directory. Depending of your platform, always add a trailing slash or backslash

```
<?php
```

```
$view->setLayoutsDir('.. /common/layouts/');
```

```
public string getLayoutsDir ()
```

Gets the current layouts sub-directory

```
public Phalcon\Mvc\View setPartialsDir (string $partialsDir)
```

Sets a partials sub-directory. Must be a directory under the views directory. Depending of your platform, always add a trailing slash or backslash

```
<?php
```

```
$view->setPartialsDir('.. /common/partials/');
```

```
public string getPartialsDir ()
```

Gets the current partials sub-directory

```
public Phalcon\Mvc\View setBasePath (string $basePath)
```

Sets base path. Depending of your platform, always add a trailing slash or backslash

```
<?php
```

```
$view->setBasePath(__DIR__ . '/');
```

```
public int getCurrentRenderLevel ()
```

Returns the render level for the view

```
public int getRenderLevel ()
```

Returns the render level for the view

```
public Phalcon\Mvc\View setRenderLevel (string $level)
```

Sets the render level for the view

```
<?php
```

```
//Render the view related to the controller only  
$this->view->setRenderLevel(View::LEVEL_LAYOUT);
```

```
public Phalcon\Mvc\View disableLevel (int/array $level)
```

Disables a specific level of rendering

```
<?php
```

```
//Render all levels except ACTION level  
$this->view->disableLevel(View::LEVEL_ACTION_VIEW);
```

```
public array getDisabledLevels ()
```

Returns an array with disabled render levels

public *Phalcon\ Mvc\ View* **setMainView** (*string* \$viewPath)

Sets default view name. Must be a file without extension in the views directory

<?php

```
//Renders as main view views-dir/base.phtml
$this->view->setMainView('base');
```

public *string* **getMainView** ()

Returns the name of the main view

public *Phalcon\ Mvc\ View* **setLayout** (*string* \$layout)

Change the layout to be used instead of using the name of the latest controller name

<?php

```
$this->view->setLayout('main');
```

public *string* **getLayout** ()

Returns the name of the main view

public *Phalcon\ Mvc\ View* **setTemplateBefore** (*string/array* \$templateBefore)

Appends template before controller layout

public *Phalcon\ Mvc\ View* **cleanTemplateBefore** ()

Resets any template before layouts

public *Phalcon\ Mvc\ View* **setTemplateAfter** (*string/array* \$templateAfter)

Appends template after controller layout

public *Phalcon\ Mvc\ View* **cleanTemplateAfter** ()

Resets any template after layouts

public *Phalcon\ Mvc\ View* **setParamToView** (*string* \$key, *mixed* \$value)

Adds parameters to views (alias of setVar)

<?php

```
$this->view->setParamToView('products', $products);
```

public *Phalcon\ Mvc\ View* **setVars** (*array* \$params, [*boolean* \$merge])

Set all the render params

<?php

```
$this->view->setVars(array('products' => $products));
```

public *Phalcon\ Mvc\ View* **setVar** (*string* \$key, *mixed* \$value)

Set a single view parameter

<?php

```
$this->view->setVar('products', $products);
```

```
public mixed getVar (string $key)
```

Returns a parameter previously set in the view

```
public array getParamsToView ()
```

Returns parameters to views

```
public string getControllerName ()
```

Gets the name of the controller rendered

```
public string getActionName ()
```

Gets the name of the action rendered

```
public array getParams ()
```

Gets extra parameters of the action rendered

```
public Phalcon\ Mvc\ View start ()
```

Starts rendering process enabling the output buffering

```
protected array _loadTemplateEngines ()
```

Loads registered template engines, if none is registered it will use Phalcon\ Mvc\ View\ Engine\ Php

```
protected _engineRender ()
```

Checks whether view exists on registered extensions and render it

```
public Phalcon\ Mvc\ View registerEngines (array $engines)
```

Register templating engines

```
<?php
```

```
$this->view->registerEngines(array(  
    ".phtml" => "Phalcon\ Mvc\ View\ Engine\ Php",  
    ".volt" => "Phalcon\ Mvc\ View\ Engine\ Volt",  
    ".mhtml" => "MyCustomEngine"  
) );
```

```
public getRegisteredEngines ()
```

Returns the registered templating engines

```
public exists (unknown $view)
```

```
...
```

```
public Phalcon\ Mvc\ View render (string $controllerName, string $actionName, [array $params])
```

Executes render process from dispatching data

```
<?php
```

```
//Shows recent posts view (app/views/posts/recent.phtml)  
$view->start()->render('posts', 'recent')->finish();
```

```
public Phalcon\ Mvc\ View pick (string/array $renderView)
```

Choose a different view to render instead of last-controller/last-action

```
<?php

class ProductsController extends Phalcon\Mvc\Controller
{

    public function saveAction()
    {

        //Do some save stuff...

        //Then show the list view
        $this->view->pick("products/list");
    }
}

public partial (string $partialPath)
Renders a partial view

<?php

//Show a partial inside another view
$this->partial('shared/footer');

<?php

//Show a partial inside another view with parameters
$this->partial('shared/footer', array('content' => $html));

public string getRender (string $controllerName, string $actionName, [array $params], [mixed $configCallback])
Perform the automatic rendering returning the output as a string

<?php

$template = $this->view->getRender('products', 'show', array('products' => $products));

public Phalcon\Mvc\View finish ()
Finishes the render process by stopping the output buffering

protected Phalcon\Cache\BackendInterface _createCache ()
Create a Phalcon\Cache based on the internal cache options

public boolean isCaching ()
Check if the component is currently caching the output content

public Phalcon\Cache\BackendInterface getCache ()
Returns the cache instance used to cache

public Phalcon\Mvc\View cache ([boolean/array $options])
Cache the actual view render to certain level

<?php

$this->view->cache(array('key' => 'my-key', 'lifetime' => 86400));
```

public *Phalcon\|Mvc\|View* **setContent** (*string* \$content)

Externally sets the view content

<?php

```
$this->view->setContent("<h1>hello</h1>");
```

public *string* **getContent** ()

Returns cached output from another view stage

public *string* **getActiveRenderPath** ()

Returns the path of the view that is currently rendered

public *Phalcon\|Mvc\|View* **disable** ()

Disables the auto-rendering process

public *Phalcon\|Mvc\|View* **enable** ()

Enables the auto-rendering process

public *bool* **isDisabled** ()

Whether automatic rendering is enabled

public *Phalcon\|Mvc\|View* **reset** ()

Resets the view component to its factory default values

public *\_\_set* (*unknown* \$property, *mixed* \$value)

Magic method to pass variables to the views

<?php

```
$this->view->products = $products;
```

public *mixed* *\_\_get* (*unknown* \$property)

Magic method to retrieve a variable passed to the view

<?php

```
echo $this->view->products;
```

public *mixed* *\_\_isset* (*unknown* \$property)

Magic method to inaccessible a variable passed to the view

<?php

```
isset($this->view->products)
```

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector) inherited from Phalcon\DI\Injectable

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** () inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager) inherited from Phalcon\DI\Injectable

Sets the event manager

`public Phalcon\\Events\ManagerInterface getEventsManager ()` inherited from `Phalcon\DI\Injectable`

Returns the internal event manager

## 2.54.208 Abstract class `Phalcon\Mvc\View\Engine`

*extends abstract class `Phalcon\DI\Injectable`*

*implements `Phalcon\Events\EventsAwareInterface`, `Phalcon\DI\InjectionAwareInterface`, `Phalcon\Mvc\View\EngineInterface`*

All the template engine adapters must inherit this class. This provides basic interfacing between the engine and the `Phalcon\Mvc\View` component.

### Methods

`public __construct (Phalcon\Mvc\ViewInterface $view, [Phalcon\DiInterface $dependencyInjector])`

`Phalcon\Mvc\View\Engine` constructor

`public array getContent ()`

Returns cached ouput on another view stage

`public string partial (string $partialPath)`

Renders a partial inside another view

`public Phalcon\Mvc\ViewInterface getView ()`

Returns the view component related to the adapter

`public setDI (Phalcon\DiInterface $dependencyInjector)` inherited from `Phalcon\DI\Injectable`

Sets the dependency injector

`public Phalcon\DiInterface getDI ()` inherited from `Phalcon\DI\Injectable`

Returns the internal dependency injector

`public setEventsManager (Phalcon\\Events\ManagerInterface $eventsManager)` inherited from `Phalcon\DI\Injectable`

Sets the event manager

`public Phalcon\\Events\ManagerInterface getEventsManager ()` inherited from `Phalcon\DI\Injectable`

Returns the internal event manager

`public __get (unknown $property)` inherited from `Phalcon\DI\Injectable`

Magic method `__get`

`abstract public render (string $path, array $params, [boolean $mustClean])` inherited from `Phalcon\Mvc\View\EngineInterface`

Renders a view using the template engine

## 2.54.209 Class Phalcon\Mvc\View\Engine\Php

*extends abstract class Phalcon\Mvc\View\Engine*

*implements Phalcon\Mvc\View\EngineInterface, Phalcon\DI\InjectionAwareInterface, Phalcon\Events\EventsAwareInterface*

Adapter to use PHP itself as templating engine

### Methods

**public render (string \$path, array \$params, [boolean \$mustClean])**

Renders a view using the template engine

**public \_\_construct (Phalcon\Mvc\ViewInterface \$view, [Phalcon\DiInterface \$dependencyInjector])**  
inherited from Phalcon\Mvc\View\Engine

Phalcon\Mvc\View\Engine constructor

**public array getContent ()** inherited from Phalcon\Mvc\View\Engine

Returns cached output on another view stage

**public string partial (string \$partialPath)** inherited from Phalcon\Mvc\View\Engine

Renders a partial inside another view

**public Phalcon\Mvc\ViewInterface getView ()** inherited from Phalcon\Mvc\View\Engine

Returns the view component related to the adapter

**public setDI (Phalcon\DiInterface \$dependencyInjector)** inherited from Phalcon\DI\Injectable

Sets the dependency injector

**public Phalcon\DiInterface getDI ()** inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

**public setEventsManager (Phalcon\Events\ManagerInterface \$eventsManager)** inherited from Phalcon\DI\Injectable

Sets the event manager

**public Phalcon\Events\ManagerInterface getEventsManager ()** inherited from Phalcon\DI\Injectable

Returns the internal event manager

**public \_\_get (unknown \$property)** inherited from Phalcon\DI\Injectable

Magic method \_\_get

## 2.54.210 Class Phalcon\Mvc\View\Engine\Volt

*extends abstract class Phalcon\Mvc\View\Engine*

*implements Phalcon\Mvc\View\EngineInterface, Phalcon\DI\InjectionAwareInterface, Phalcon\Events\EventsAwareInterface*

Designer friendly and fast template engine for PHP written in C

## Methods

**public `setOptions`** (*array \$options*)

Set Volt's options

**public `array getOptions`** ()

Return Volt's options

**public `Phalcon\ Mvc\ View\ Engine\ Volt\ Compiler getCompiler`** ()

Returns the Volt's compiler

**public `render`** (*unknown \$path, array \$params, [boolean \$mustClean]*)

Renders a view using the template engine

**public `int length`** (*mixed \$item*)

Length filter. If an array/object is passed a count is performed otherwise a strlen(mb\_strlen)

**public `boolean isIncluded`** (*mixed \$needle, mixed \$haystack*)

Checks if the needle is included in the haystack

**public `string convertEncoding`** (*string \$text, string \$from, string \$to*)

Performs a string conversion

**public `slice`** (*mixed \$value, unknown \$start, [unknown \$end]*)

Extracts a slice from a string/array/traversable object value

**public `array sort`** (*array \$value*)

Sorts an array

**public `__construct`** (*Phalcon\ Mvc\ ViewInterface \$view, [Phalcon\ DiInterface \$dependencyInjector]*)  
inherited from Phalcon\ Mvc\ View\ Engine

Phalcon\ Mvc\ View\ Engine constructor

**public `array getContent`** () inherited from Phalcon\ Mvc\ View\ Engine

Returns cached output on another view stage

**public `string partial`** (*string \$partialPath*) inherited from Phalcon\ Mvc\ View\ Engine

Renders a partial inside another view

**public `Phalcon\ Mvc\ ViewInterface getView`** () inherited from Phalcon\ Mvc\ View\ Engine

Returns the view component related to the adapter

**public `setDI`** (*Phalcon\ DiInterface \$dependencyInjector*) inherited from Phalcon\ DI\ Injectable

Sets the dependency injector

**public `Phalcon\ DiInterface getDI`** () inherited from Phalcon\ DI\ Injectable

Returns the internal dependency injector

**public `setEventsManager`** (*Phalcon\ Events\ ManagerInterface \$eventsManager*) inherited from Phalcon\ DI\ Injectable

Sets the event manager

**public `Phalcon\ Events\ ManagerInterface getEventsManager`** () inherited from Phalcon\ DI\ Injectable

Returns the internal event manager

public **\_\_get** (*unknown* \$property) inherited from Phalcon\DI\Injectable

Magic method **\_\_get**

### 2.54.211 Class Phalcon\Mvc\View\Engine\Volt\Compiler

*implements Phalcon\DI\InjectionAwareInterface*

This class reads and compiles Volt templates into PHP plain code

```
<?php  
  
$compiler = new \Phalcon\Mvc\View\Engine\Volt\Compiler();  
  
$compiler->compile('views/partials/header.volt');  
  
require $compiler->getCompiledTemplatePath();
```

#### Methods

public **\_\_construct** ([*Phalcon\ Mvc\ ViewInterface* \$view])

public **setDI** (*Phalcon\ DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\ DiInterface* **getDI** ()

Returns the internal dependency injector

public **setOptions** (*array* \$options)

Sets the compiler options

public **setOption** (*string* \$option, *string* \$value)

Sets a single compiler option

public *string* **getOption** (*string* \$option)

Returns a compiler's option

public *array* **getOptions** ()

Returns the compiler options

public *mixed* **fireExtensionEvent** (*string* \$name, [*array* \$arguments])

Fires an event to registered extensions

public *Phalcon\ Mvc\ View\ Engine\ Volt\ Compiler* **addExtension** (*object* \$extension)

Registers a Volt's extension

public *array* **getExtensions** ()

Returns the list of extensions registered in Volt

public *Phalcon\ Mvc\ View\ Engine\ Volt\ Compiler* **addFunction** (*string* \$name, *Closure/string* \$definition)

Register a new function in the compiler

public *array* **getFunctions** ()

Register the user registered functions

`public Phalcon\|Mvc\|View\|Engine\|Volt\|Compiler addFilter (string $name, Closure/string $definition)`

Register a new filter in the compiler

`public array getFilters ()`

Register the user registered filters

`public Phalcon\|Mvc\|View\|Engine\|Volt\|Compiler setUniquePrefix (string $prefix)`

Set a unique prefix to be used as prefix for compiled variables

`public string getUniquePrefix ()`

Return a unique prefix to be used as prefix for compiled variables and contexts

`public string attributeReader (array $expr)`

Resolves attribute reading

`public string functionCall (array $expr)`

Resolves function intermediate code into PHP function calls

`public string resolveTest (array $test, string $left)`

Resolves filter intermediate code into a valid PHP expression

`protected string resolveFilter ()`

Resolves filter intermediate code into PHP function calls

`public string expression (array $expr)`

Resolves an expression node in an AST volt tree

`protected string/array _statementListOrExtends ()`

Compiles a block of statements

`public string compileForeach (array $statement, [boolean $extendsMode])`

Compiles a ‘foreach’ intermediate code representation into plain PHP code

`public string compileForElse ()`

Generates a ‘forelse’ PHP code

`public string compileIf (array $statement, [boolean $extendsMode])`

Compiles a ‘if’ statement returning PHP code

`public string compileElseIf (array $statement)`

Compiles a ‘elseif’ statement returning PHP code

`public string compileCache (array $statement, [boolean $extendsMode])`

Compiles a ‘cache’ statement returning PHP code

`public string compileEcho (array $statement)`

Compiles a ‘{{ ‘ }}’ statement returning PHP code

`public string compileInclude (array $statement)`

Compiles a ‘include’ statement returning PHP code

`public string compileSet (array $statement)`

Compiles a ‘set’ statement returning PHP code

public *string* **compileDo** (*array* \$statement)

Compiles a ‘do’ statement returning PHP code

public *string* **compileReturn** (*array* \$statement)

Compiles a ‘return’ statement returning PHP code

public *string* **compileAutoEscape** (*array* \$statement, *boolean* \$extendsMode)

Compiles a ‘autoescape’ statement returning PHP code

public *string* **compileMacro** (*array* \$statement, *boolean* \$extendsMode)

Compiles macros

public *string* **compileCall** ()

Compiles calls to macros

protected *string* **\_statementList** ()

Traverses a statement list compiling each of its nodes

protected *string* **\_compileSource** ()

Compiles a Volt source code returning a PHP plain version

public *string* **compileString** (*string* \$viewCode, [*boolean* \$extendsMode])

Compiles a template into a string

<?php

```
echo $compiler->compileString('{{ "hello world" }}');
```

public *string/array* **compileFile** (*string* \$path, *string* \$compiledPath, [*boolean* \$extendsMode])

Compiles a template into a file forcing the destination path

<?php

```
$compiler->compile('views/layouts/main.volt', 'views/layouts/main.volt.php');
```

public *string/array* **compile** (*string* \$templatePath, [*boolean* \$extendsMode])

Compiles a template into a file applying the compiler options This method does not return the compiled path if the template was not compiled

<?php

```
$compiler->compile('views/layouts/main.volt');
require $compiler->getCompiledTemplatePath();
```

public *string* **getTemplatePath** ()

Returns the path that is currently being compiled

public *string* **getCompiledTemplatePath** ()

Returns the path to the last compiled template

public *array* **parse** (*string* \$viewCode)

Parses a Volt template returning its intermediate representation

```
<?php
print_r($compiler->parse('{{ 3 + 2 }}'));
```

## 2.54.212 Class Phalcon\Mvc\View\Exception

*extends class Phalcon\Exception*

Class for exceptions thrown by Phalcon\Mvc\View

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

## 2.54.213 Class Phalcon\Mvc\View\Simple

*extends abstract class Phalcon\DI\Injectable*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\DI\InjectionAwareInterface*

This component allows to render views without hierarchical levels

```
<?php
```

```
$view = new Phalcon\Mvc\View\Simple();
echo $view->render('templates/my-view', array('content' => $html));
```

## Methods

public **\_\_construct** ([array \$options])

Phalcon\Mvc\View constructor

public **setViewsDir** (string \$viewsDir)

Sets views directory. Depending of your platform, always add a trailing slash or backslash

public **getViewsDir** ()

Gets views directory

public **registerEngines** (array \$engines)

Register templating engines

```
<?php
```

```
$this->view->registerEngines(array(
    ".phtml" => "Phalcon\Mvc\View\Engine\Php",
    ".volt" => "Phalcon\Mvc\View\Engine\Volt",
    ".mhtml" => "MyCustomEngine"
));
```

public **getRegisteredEngines** ()

Returns the registered templating engines

protected **\_loadTemplateEngines** ()

Loads registered template engines, if none is registered it will use Phalcon\Mvc\View\Engine\Php

protected **\_internalRender** ()

Tries to render the view with every engine registered in the component

public **render** (string \$path, [array \$params])

Renders a view

public **partial** (string \$partialPath, [array \$params])

Renders a partial view

```
<?php
```

```
//Show a partial inside another view
$this->partial('shared/footer');
```

```
<?php
```

```
//Show a partial inside another view with parameters
$this->partial('shared/footer', array('content' => $html));
```

public *Phalcon\ Mvc\ View\ Simple* **setCacheOptions** (*array* \$options)

Sets the cache options

public *array* **getCacheOptions** ()

Returns the cache options

protected *Phalcon\ Cache\ BackendInterface* **\_createCache** ()

Create a Phalcon\Cache based on the internal cache options

public *Phalcon\ Cache\ BackendInterface* **getCache** ()

Returns the cache instance used to cache

public *Phalcon\ Mvc\ View\ Simple* **cache** ([*boolean/array* \$options])

Cache the actual view render to certain level

<?php

```
$this->view->cache(array('key' => 'my-key', 'lifetime' => 86400));
```

public *Phalcon\ Mvc\ View\ Simple* **setParamToView** (*string* \$key, *mixed* \$value)

Adds parameters to views (alias of setVar)

<?php

```
$this->view->setParamToView('products', $products);
```

public *Phalcon\ Mvc\ View\ Simple* **setVars** (*array* \$params, [*boolean* \$merge])

Set all the render params

<?php

```
$this->view->setVars(array('products' => $products));
```

public *Phalcon\ Mvc\ View\ Simple* **setVar** (*string* \$key, *mixed* \$value)

Set a single view parameter

<?php

```
$this->view->setVar('products', $products);
```

public *mixed* **getVar** (*string* \$key)

Returns a parameter previously set in the view

public *array* **getParamsToView** ()

Returns parameters to views

public *Phalcon\ Mvc\ View\ Simple* **setContent** (*string* \$content)

Externally sets the view content

<?php

```
$this->view->setContent("<h1>hello</h1>");
```

```
public string getContent ()  
Returns cached output from another view stage  
public string getActiveRenderPath ()  
Returns the path of the view that is currently rendered  
public __set (string $key, mixed $value)  
Magic method to pass variables to the views  
<?php  
  
$this->view->products = $products;  
  
public mixed __get (string $key)  
Magic method to retrieve a variable passed to the view  
<?php  
  
echo $this->view->products;  
  
public setDI (Phalcon\DiInterface $dependencyInjector) inherited from Phalcon\DI\Injectable  
Sets the dependency injector  
public Phalcon\DiInterface getDI () inherited from Phalcon\DI\Injectable  
Returns the internal dependency injector  
public setEventsManager (Phalcon\Events\ManagerInterface $eventsManager) inherited from Phalcon\DI\Injectable  
Sets the event manager  
public Phalcon\Events\ManagerInterface getEventsManager () inherited from Phalcon\DI\Injectable  
Returns the internal event manager
```

## 2.54.214 Class Phalcon\Paginator\Adapter\Model

*implements Phalcon\Paginator\AdapterInterface*

This adapter allows to paginate data using a Phalcon\Mvc\Model resultset as base

### Methods

```
public __construct (array $config)  
Phalcon\Paginator\Adapter\Model constructor  
public setCurrentPage (int $page)  
Set the current page number  
public stdClass getPaginate ()  
Returns a slice of the resultset to show in the pagination
```

## 2.54.215 Class Phalcon\Paginator\Adapter\NativeArray

*implements Phalcon\Paginator\AdapterInterface*

Pagination using a PHP array as source of data

```
<?php

$paginator = new \Phalcon\Paginator\Adapter\Model(
    array(
        "data" => array(
            array('id' => 1, 'name' => 'Artichoke'),
            array('id' => 2, 'name' => 'Carrots'),
            array('id' => 3, 'name' => 'Beet'),
            array('id' => 4, 'name' => 'Lettuce'),
            array('id' => 5, 'name' => '')
        ),
        "limit" => 2,
        "page" => $currentPage
    )
);
;
```

### Methods

public **\_\_construct** (*array \$config*)

Phalcon\Paginator\Adapter\NativeArray constructor

public **setCurrentPage** (*int \$page*)

Set the current page number

public *stdClass* **getPaginate** ()

Returns a slice of the resultset to show in the pagination

## 2.54.216 Class Phalcon\Paginator\Adapter\QueryBuilder

*implements Phalcon\Paginator\AdapterInterface*

Pagination using a PHQL query builder as source of data

```
<?php

$builder = $this->modelsManager->createBuilder()
    ->columns('id, name')
    ->from('Robots')
    ->orderBy('name');

$paginator = new Phalcon\Paginator\Adapter\QueryBuilder(array(
    "builder" => $builder,
    "limit"=> 20,
    "page" => 1
));
;
```

### Methods

public **\_\_construct** (*array \$config*)

public *stdClass* **getPaginate** ()

Returns a slice of the resultset to show in the pagination

public *Phalcon\Paginator\Adapter\QueryBuilder* \$this Fluent interface **setLimit** (*int* \$limit)

Set current rows limit

public *int* \$limit **getLimit** ()

Get current rows limit

public **setCurrentPage** (*int* \$page)

Set current page number

public **getCurrentPage** ()

Get current page number

public *Phalcon\Paginator\Adapter\QueryBuilder* \$this Fluent interface **setQueryBuilder** (*unknown* \$queryBuilder)

Set query builder object

public *Phalcon\Mvc\Model\Query\BuilderInterface* \$builder **getQueryBuilder** ()

Get query builder object

## 2.54.217 Class Phalcon\Paginator\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Paginator will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

`final public Exception getTraceAsString ()` inherited from `Exception`

Gets the stack trace as a string

`public string __toString ()` inherited from `Exception`

String representation of the exception

## 2.54.218 Class Phalcon\Queue\Beanstalk

Class to access the beanstalk queue service. Partially implements the protocol version 1.2

### Methods

`public __construct ([array $options])`

`public connect ()`

...

`public string/boolean put (string $data, [array $options])`

Inserts jobs into the queue

`public boolean/Phalcon\Queue\Beanstalk\Job reserve ([unknown $timeout])`

Reserves a job in the queue

`public string/boolean choose (string $tube)`

Change the active tube. By default the tube is ‘default’

`public string/boolean watch (string $tube)`

Change the active tube. By default the tube is ‘default’

`public boolean/Phalcon\Queue\Beanstalk\Job peekReady ()`

Inspect the next ready job.

`public boolean/Phalcon\Queue\Beanstalk\Job peekDelayed ()`

Return the delayed job with the shortest delay left

`public boolean/Phalcon\Queue\Beanstalk\Job peekBuried ()`

Return the next job in the list of buried jobs

`protected array readStatus ()`

Reads the latest status from the Beanstalkd server

`public string/boolean Data or ‘false’ on error. read ([unknown $length])`

Reads a packet from the socket. Prior to reading from the socket will check for availability of the connection.

`protected integer/boolean write ()`

Writes data to the socket. Performs a connection if none is available

`public boolean disconnect ()`

Closes the connection to the beanstalk server.

`public __sleep ()`

```
...
public __wakeup ()
...

```

## 2.54.219 Class Phalcon\Queue\Beanstalk\Job

Represents a job in a beanstalk queue

### Methods

public **\_\_construct** (*Phalcon\Queue|Beanstalk* \$queue, *string* \$id, *mixed* \$body)

public *string* **getId** ()

Returns the job id

public *mixed* **getBody** ()

Returns the job body

public *boolean* **delete** ()

Removes a job from the server entirely

public *boolean* **release** ()

The release command puts a reserved job back into the ready queue (and marks its state as “ready”) to be run by any client. It is normally used when the job fails because of a transitory error.

public *boolean* **bury** ()

The bury command puts a job into the “buried” state. Buried jobs are put into a FIFO linked list and will not be touched by the server again until a client kicks them with the “kick” command.

public *boolean* **touch** ()

The bury command puts a job into the “buried” state. Buried jobs are put into a FIFO linked list and will not be touched by the server again until a client kicks them with the “kick” command.

public *boolean* **kick** ()

Move the job to the ready queue if it is delayed or buried.

public **\_\_wakeup** ()

...

## 2.54.220 Final class Phalcon\Registry

*implements* ArrayAccess, Iterator, Traversable, Serializable, Countable, JsonSerializable

A registry is a container for storing objects and values in the application space. By storing the value in a registry, the same object is always available throughout your application.

```
<?php
$registry = new \Phalcon\Registry();

// Set value
$registry->something = 'something';

```

```
// or
$registry['something'] = 'something';

// Get value
$value = $registry->something;
// or
$value = $registry['something'];

// Check if the key exists
$exists = isset($registry->something);
// or
$exists = isset($registry['something']);

// Unset
unset($registry->something);
// or
unset($registry['something']);
```

In addition to `ArrayAccess`, `Phalcon\\Registry` also `implements Countable` (`count($registry)` will return the number of elements).

## Methods

```
public __get (unknown $property)
...
public __set (unknown $property, unknown $value)
...
public __isset (unknown $property)
...
public __unset (unknown $property)
...
public __call (unknown $method, [unknown $arguments])
...
public count ()
...
public offsetGet (unknown $property)
...
public offsetSet (unknown $property, unknown $value)
...
public offsetUnset (unknown $property)
...
public offsetExists (unknown $property)
...
public current ()
```

```
...
public key ()
...
public next ()
...
public rewind ()
...
public valid ()
...
public jsonSerialize ()
...
public serialize ()
...
public unserialize ([unknown $serialized])
...
private __wakeup ()
```

## 2.54.221 Class Phalcon\Security

*implements Phalcon\DI\InjectionAwareInterface*

This component provides a set of functions to improve the security in Phalcon applications

<?php

```
$login = $this->request->getPost('login');
$password = $this->request->getPost('password');

$user = Users::findFirstByLogin($login);
if ($user) {
    if ($this->security->checkHash($password, $user->password)) {
        //The password is valid
    }
}
```

### Constants

```
integer CRYPT_DEFAULT
integer CRYPT_STD_DES
integer CRYPT_EXT_DES
integer CRYPT_MD5
integer CRYPT_BLOWFISH
```

```
integer CRYPT_BLOWFISH_X  
integer CRYPT_BLOWFISH_Y  
integer CRYPT_SHA256  
integer CRYPT_SHA512
```

## Methods

public **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector

public *Phalcon\DiInterface* **getDI** ()

Returns the internal dependency injector

public **setRandomBytes** (*string* \$randomBytes)

Sets a number of bytes to be generated by the openssl pseudo random generator

public *string* **getRandomBytes** ()

Returns a number of bytes to be generated by the openssl pseudo random generator

public **setWorkFactor** (*int* \$workFactor)

Sets the default working factor for bcrypts password's salts

public *int* **getWorkFactor** ()

Returns the default working factor for bcrypts password's salts

public *string* **getSaltBytes** ()

Generate a >22-length pseudo random string to be used as salt for passwords

public *string* **hash** (*string* \$password, [*int* \$workFactor])

Creates a password hash using bcrypt with a pseudo random salt

public *boolean* **checkHash** (*string* \$password, *string* \$passwordHash, [*int* \$maxLength])

Checks a plain text password and its hash version to check if the password matches

public *boolean* **isLegacyHash** (*string* \$passwordHash)

Checks if a password hash is a valid bcrypt's hash

public *string* **getTokenKey** ([*int* \$numberBytes])

Generates a pseudo random token key to be used as input's name in a CSRF check

public *string* **getToken** ([*int* \$numberBytes])

Generates a pseudo random token value to be used as input's value in a CSRF check

public *boolean* **checkToken** ([*string* \$tokenKey], [*string* \$tokenValue])

Check if the CSRF token sent in the request is the same that the current in session

public *string* **getSessionToken** ()

Returns the value of the CSRF token in session

public static **computeHmac** (*unknown* \$data, *unknown* \$key, *unknown* \$algo, [*unknown* \$raw])

*string* \Phalcon\Security::computeHmac(*string* \$data, *string* \$key, *string* \$algo, *bool* \$raw = false)

public static *string* *The derived key* **deriveKey** (*unknown \$password*, *unknown \$salt*, [*unknown \$hash*], [*unknown \$iterations*], [*unknown \$size*])

Derives a key from the given password (PBKDF2).

public static **pbkdf2** (*unknown \$password*, *unknown \$salt*, [*unknown \$hash*], [*unknown \$iterations*], [*unknown \$size*])

public **getDefaultHash** ()

Returns the default hash

public **setDefaultHash** (*unknown \$hash*)

Sets the default hash

## 2.54.222 Class Phalcon\Security\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Security will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

## 2.54.223 Abstract class Phalcon\Session\Adapter

*implements Phalcon\Session\AdapterInterface*, Countable, IteratorAggregate, Traversable, ArrayAccess  
Base class for Phalcon\Session adapters

### Methods

public **\_\_construct** ([array \$options])

Phalcon\Session\Adapter constructor

public **\_\_destruct** ()

...

public **boolean start** ()

Starts the session (if headers are already sent the session will not be started)

public **setOptions** (array \$options)

Sets session's options

<?php

```
$session->setOptions(array(
    'uniqueId' => 'my-private-app',
));
```

public **array getOptions** ()

Get internal options

public **mixed get** (string \$index, [mixed \$defaultValue])

Gets a session variable from an application context

public **set** (string \$index, string \$value)

Sets a session variable in an application context

<?php

```
$session->set('auth', 'yes');
```

public **boolean has** (string \$index)

Check whether a session variable is set in an application context

<?php

```
var_dump($session->has('auth'));
```

public **remove** (string \$index)

Removes a session variable from an application context

<?php

```
$session->remove('auth');
```

public *string* getId ()

Returns active session id

<?php

echo \$session->getId();

public *boolean* isStarted ()

Check whether the session has been started

<?php

var\_dump(\$session->isStarted());

public *boolean* destroy ([*unknown* \$session\_id])

Destroys the active session

<?php

var\_dump(\$session->destroy());

public \_\_get (*unknown* \$property)

...

public \_\_set (*unknown* \$property, *unknown* \$value)

...

public \_\_isset (*unknown* \$property)

...

public \_\_unset (*unknown* \$property)

...

public offsetGet (*unknown* \$property)

...

public offsetSet (*unknown* \$property, *unknown* \$value)

...

public offsetExists (*unknown* \$property)

...

public offsetUnset (*unknown* \$property)

...

public count ()

...

public getIterator ()

...

public setId (*unknown* \$sid)

Set the current session id

```
<?php
$session->setId($id);
```

## 2.54.224 Class Phalcon\Session\Adapter\Files

*extends abstract class Phalcon\Session\Adapter*

*implements ArrayAccess, Traversable, IteratorAggregate, Countable, Phalcon\Session\AdapterInterface*

This adapter store sessions in plain files

```
<?php
$session = new Phalcon\Session\Adapter\Files(array(
    'uniqueId' => 'my-private-app'
));

$session->start();

$session->set('var', 'some-value');

echo $session->get('var');
```

### Methods

public **\_\_construct** ([array \$options]) inherited from Phalcon\Session\Adapter

Phalcon\Session\Adapter constructor

public **\_\_destruct** () inherited from Phalcon\Session\Adapter

...

public **boolean start** () inherited from Phalcon\Session\Adapter

Starts the session (if headers are already sent the session will not be started)

public **setOptions** (array \$options) inherited from Phalcon\Session\Adapter

Sets session's options

```
<?php
$session->setOptions(array(
    'uniqueId' => 'my-private-app'
));
```

public **array getOptions** () inherited from Phalcon\Session\Adapter

Get internal options

public **mixed get** (string \$index, [mixed \$defaultValue]) inherited from Phalcon\Session\Adapter

Gets a session variable from an application context

public **set** (string \$index, string \$value) inherited from Phalcon\Session\Adapter

Sets a session variable in an application context

```
<?php
```

```
$session->set('auth', 'yes');
```

public **boolean has** (*string \$index*) inherited from Phalcon\Session\Adapter

Check whether a session variable is set in an application context

```
<?php
```

```
var_dump($session->has('auth'));
```

public **remove** (*string \$index*) inherited from Phalcon\Session\Adapter

Removes a session variable from an application context

```
<?php
```

```
$session->remove('auth');
```

public **string getId** () inherited from Phalcon\Session\Adapter

Returns active session id

```
<?php
```

```
echo $session->getId();
```

public **boolean isStarted** () inherited from Phalcon\Session\Adapter

Check whether the session has been started

```
<?php
```

```
var_dump($session->isStarted());
```

public **boolean destroy** ([*unknown \$session\_id*]) inherited from Phalcon\Session\Adapter

Destroys the active session

```
<?php
```

```
var_dump($session->destroy());
```

public **\_\_get** (*unknown \$property*) inherited from Phalcon\Session\Adapter

...

public **\_\_set** (*unknown \$property, unknown \$value*) inherited from Phalcon\Session\Adapter

...

public **\_\_isset** (*unknown \$property*) inherited from Phalcon\Session\Adapter

...

public **\_\_unset** (*unknown \$property*) inherited from Phalcon\Session\Adapter

...

public **offsetGet** (*unknown \$property*) inherited from Phalcon\Session\Adapter

...

public **offsetSet** (*unknown \$property, unknown \$value*) inherited from Phalcon\Session\Adapter

...

public **offsetExists** (*unknown* \$property) inherited from Phalcon\Session\Adapter

...

public **offsetUnset** (*unknown* \$property) inherited from Phalcon\Session\Adapter

...

public **count** () inherited from Phalcon\Session\Adapter

...

public **getIterator** () inherited from Phalcon\Session\Adapter

...

public **setId** (*unknown* \$sid) inherited from Phalcon\Session\Adapter

Set the current session id

```
<?php
$session->setId($id);
```

## 2.54.225 Class Phalcon\Session\Bag

*implements* [Phalcon\DI\InjectionAwareInterface](#), [Phalcon\Session\BagInterface](#), IteratorAggregate, Traversable, ArrayAccess, Countable

This component helps to separate session data into “namespaces”. Working by this way you can easily create groups of session variables into the application

```
<?php
$user = new \Phalcon\Session\Bag('user');
$user->name = "Kimbra Johnson";
$user->age = 22;
```

### Methods

public **\_\_construct** (*string* \$name)

Phalcon\Session\Bag constructor

public **setDI** ([Phalcon\DiInterface](#) \$dependencyInjector)

Sets the DependencyInjector container

public [Phalcon\DiInterface](#) **getDI** ()

Returns the DependencyInjector container

public **initialize** ()

Initializes the session bag. This method must not be called directly, the class calls it when its internal data is accessed

public **destroy** ()

Destroys the session bag

<?php

\$user->destroy();

public **set** (*string* \$property, *string* \$value)

Sets a value in the session bag

<?php

\$user->set('name', 'Kimbra');

public *mixed* **get** (*string* \$property, [*string* \$defaultValue])

Obtains a value from the session bag optionally setting a default value

<?php

echo \$user->get('name', 'Kimbra');

public *boolean* **has** (*string* \$property)

Check whether a property is defined in the internal bag

<?php

var\_dump(\$user->has('name'));

public *boolean* **remove** (*string* \$property)

Removes a property from the internal bag

<?php

\$user->remove('name');

public **getIterator** ()

...

public *string* **\_\_get** (*string* \$property)

Magic getter to obtain values from the session bag.

<?php

echo \$user->name;

public **\_\_set** (*string* \$property, *string* \$value)

Magic setter to assign values to the session bag. Alias for Phalcon\Session\Bag::set()

<?php

\$user->name = "Kimbra";

public *boolean* **\_\_isset** (*string* \$property)

Magic isset to check whether a property is defined in the bag. Alias for Phalcon\Session\Bag::has()

<?php

var\_dump(isset(\$user['name']));

```

public boolean __unset (string $property)
Magic unset to remove items using the property syntax. Alias for Phalcon\Session\Bag::remove()
<?php
unset($user['name']);

public offsetGet (unknown $property)
...
public offsetSet (unknown $property, unknown $value)
...
public offsetExists (unknown $property)
...
public offsetUnset (unknown $property)
...
public count ()
...

```

## 2.54.226 Class Phalcon\Session\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Session will use this class

### Methods

final private *Exception* \_\_clone () inherited from Exception

Clone the exception

public \_\_construct ([string \$message], [int \$code], [Exception \$previous]) inherited from Exception

Exception constructor

final public string getMessage () inherited from Exception

Gets the Exception message

final public int getCode () inherited from Exception

Gets the Exception code

final public string getFile () inherited from Exception

Gets the file in which the exception occurred

final public int getLine () inherited from Exception

Gets the line in which the exception occurred

final public array getTrace () inherited from Exception

Gets the stack trace

final public *Exception* getPrevious () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from *Exception*

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from *Exception*

String representation of the exception

## 2.54.227 Class Phalcon\Tag

Phalcon\Tag is designed to simplify building of HTML tags. It provides a set of helpers to generate HTML in a dynamic way. This component is an abstract class that you can extend to add more helpers.

### Constants

*integer* **HTML32**

*integer* **HTML401\_STRICT**

*integer* **HTML401\_TRANSITIONAL**

*integer* **HTML401\_FRAMESET**

*integer* **HTML5**

*integer* **XHTML10\_STRICT**

*integer* **XHTML10\_TRANSITIONAL**

*integer* **XHTML10\_FRAMESET**

*integer* **XHTML11**

*integer* **XHTML20**

*integer* **XHTML5**

### Methods

public static **setDI** (*Phalcon\DiInterface* \$dependencyInjector)

Sets the dependency injector container.

public static *Phalcon\DiInterface* **getDI** ()

Internally gets the dependency injector

public static *Phalcon\Mvc\UrlInterface* **getUrlService** ()

Return a URL service from the default DI

public static *Phalcon\EscaperInterface* **getEscaperService** ()

Returns an Escaper service from the default DI

public static *bool* **getAutoescape** ()

Get current autoescape mode

public static **setAutoescape** (*boolean* \$autoescape)

Set autoescape mode in generated html

public static **setDefault** (*string \$id, string \$value*)

Assigns default values to generated tags by helpers

<?php

```
//Assigning "peter" to "name" component
Phalcon\Tag::setDefault("name", "peter");

//Later in the view
echo Phalcon\Tag::textField("name"); //Will have the value "peter" by default
```

public static **setDefaults** (*array \$values*)

Assigns default values to generated tags by helpers

<?php

```
//Assigning "peter" to "name" component
Phalcon\Tag::setDefaults(array("name" => "peter"));

//Later in the view
echo Phalcon\Tag::textField("name"); //Will have the value "peter" by default
```

public static **displayTo** (*string \$id, string \$value*)

Alias of Phalcon\Tag::setDefault

public static *boolean* **hasValue** (*string \$name*)

Check if a helper has a default value set using Phalcon\Tag::setDefault or value from \$\_POST

public static *mixed* **getValue** (*string \$name, [array \$params]*)

Every helper calls this function to check whether a component has a predefined value using Phalcon\Tag::setDefault or value from \$\_POST

public static **resetInput** ()

Resets the request and internal values to avoid those fields will have any default value

public static *string* **linkTo** (*array/string \$parameters, [string \$text]*)

Builds a HTML A tag using framework conventions

<?php

```
echo Phalcon\Tag::linkTo('signup/register', 'Register Here!');
echo Phalcon\Tag::linkTo(array('signup/register', 'Register Here!'));
echo Phalcon\Tag::linkTo(array('signup/register', 'Register Here!', 'class' => 'btn-primary'));
echo Phalcon\Tag::linkTo('http://phalconphp.com/', 'Google', FALSE);
echo Phalcon\Tag::linkTo(array('http://phalconphp.com/', 'Phalcon Home', FALSE));
echo Phalcon\Tag::linkTo(array('http://phalconphp.com/', 'Phalcon Home', 'local' =>FALSE));
```

protected static *string* **\_inputField** ()

Builds generic INPUT tags

protected static *string* **\_inputFieldChecked** ()

Builds INPUT tags that implements the checked attribute

public static *string* **colorField** (*array \$parameters*)

Builds a HTML input[type="color"] tag

```
public static string textField (array $parameters)
Builds a HTML input[type="text"] tag

<?php

echo Phalcon\Tag::textField(array("name", "size" => 30));

public static string numericField (array $parameters)
Builds a HTML input[type="number"] tag

<?php

echo Phalcon\Tag::numericField(array("price", "min" => "1", "max" => "5"));

public static string rangeField (array $parameters)
Builds a HTML input[type="range"] tag

public static string emailField (array $parameters)
Builds a HTML input[type="email"] tag

<?php

echo Phalcon\Tag::emailField("email");

public static string dateField (array $parameters)
Builds a HTML input[type="date"] tag

<?php

echo Phalcon\Tag::dateField(array("born", "value" => "14-12-1980"))

public static string dateTimeField (array $parameters)
Builds a HTML input[type="datetime"] tag

public static string dateTimeLocalField (array $parameters)
Builds a HTML input[type="datetime-local"] tag

public static string monthField (array $parameters)
Builds a HTML input[type="month"] tag

public static string timeField (array $parameters)
Builds a HTML input[type="time"] tag

public static string weekField (array $parameters)
Builds a HTML input[type="week"] tag

public static string passwordField (array $parameters)
Builds a HTML input[type="password"] tag

<?php

echo Phalcon\Tag::passwordField(array("name", "size" => 30));
```

public static *string* **hiddenField** (*array* \$parameters)

Builds a HTML input[type="hidden"] tag

<?php

```
echo Phalcon\Tag::hiddenField(array("name", "value" => "mike"));
```

public static *string* **searchField** (*array* \$parameters)

Builds a HTML input[type="search"] tag

public static *string* **telField** (*array* \$parameters)

Builds a HTML input[type="tel"] tag

public static *string* **urlField** (*array* \$parameters)

Builds a HTML input[type="url"] tag

public static *string* **fileField** (*array* \$parameters)

Builds a HTML input[type="file"] tag

<?php

```
echo Phalcon\Tag::fileField("file");
```

public static *string* **checkField** (*array* \$parameters)

Builds a HTML input[type="checkbox"] tag

<?php

```
echo Phalcon\Tag::checkField(array("terms", "value" => "Y"));
```

public static *string* **radioField** (*array* \$parameters)

Builds a HTML input[type="radio"] tag

<?php

```
echo Phalcon\Tag::radioField(array("wheather", "value" => "hot"));
```

Volt syntax:

<?php

```
{{ radio_field('Save') }}
```

public static *string* **imageInput** (*array* \$parameters)

Builds a HTML input[type="image"] tag

<?php

```
echo Phalcon\Tag::imageInput(array("src" => "/img/button.png"));
```

Volt syntax:

<?php

```
{{ image_input('src': '/img/button.png') }}
```

public static *string* **submitButton** (*array* \$parameters)

Builds a HTML input[type="submit"] tag

<?php

```
echo Phalcon\Tag::submitButton("Save")
```

Volt syntax:

<?php

```
{f submit_button('Save') }}
```

public static *string* **selectStatic** (*array* \$parameters, [*array* \$data])

Builds a HTML SELECT tag using a PHP array for options

<?php

```
echo Phalcon\Tag::selectStatic("status", array("A" => "Active", "I" => "Inactive"))
```

public static *string* **select** (*array* \$parameters, [*array* \$data])

Builds a HTML SELECT tag using a Phalcon\Mvc\Model resultset as options

<?php

```
echo Phalcon\Tag::select(array(
    "robotId",
    Robots::find("type = 'mechanical'"),
    "using" => array("id", "name")
));
```

Volt syntax:

<?php

```
{f select("robotId", robots, "using": ["id", "name"]) }}
```

public static *string* **textArea** (*array* \$parameters)

Builds a HTML TEXTAREA tag

<?php

```
echo Phalcon\Tag::textArea(array("comments", "cols" => 10, "rows" => 4))
```

Volt syntax:

<?php

```
{f text_area("comments", "cols": 10, "rows": 4) }
```

public static *string* **form** ([*array* \$parameters])

Builds a HTML FORM tag

<?php

```
echo Phalcon\Tag::form("posts/save");
echo Phalcon\Tag::form(array("posts/save", "method" => "post"));
```

Volt syntax:

```
<?php
{{ form("posts/save") }}
{{ form("posts/save", "method": "post") }}
```

public static *string* **endForm** ()

Builds a HTML close FORM tag

public static **setTitle** (*string* \$title)

Set the title of view content

```
<?php
```

```
Phalcon\Tag::setTitle('Welcome to my Page');
```

public static **setTitleSeparator** (*unknown* \$separator)

Set the title separator of view content

```
<?php
```

```
Phalcon\Tag::setTitleSeparator('-');
```

public static **appendTitle** (*string* \$title)

Appends a text to current document title

public static **prependTitle** (*string* \$title)

Prepends a text to current document title

public static *string* **getTitle** ([*unknown* \$tags])

Gets the current document title

```
<?php
```

```
echo Phalcon\Tag::getTitle();
```

```
<?php
```

```
{}{ get_title() }}
```

public static *string* **getTitleSeparator** ()

Gets the current document title separator

```
<?php
```

```
echo Phalcon\Tag::getTitleSeparator();
```

```
<?php
```

```
{}{ get_title_separator() }}
```

public static *string* **stylesheetLink** ([*array* \$parameters], [*boolean* \$local])

Builds a LINK[rel="stylesheet"] tag

```
<?php
```

```
echo Phalcon\Tag::stylesheetLink("http://fonts.googleapis.com/css?family=Rosario", false);
echo Phalcon\Tag::stylesheetLink("css/style.css");
```

Volt Syntax:

```
<?php
```

```
{{ stylesheet_link("http://fonts.googleapis.com/css?family=Rosario", false) }}
{{ stylesheet_link("css/style.css") }}
```

public static *string* **javascriptInclude** ([*array* \$parameters], [*boolean* \$local])

Builds a SCRIPT[type="javascript"] tag

```
<?php
```

```
echo Phalcon\Tag::javascriptInclude("http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js", false);
echo Phalcon\Tag::javascriptInclude("javascript/jquery.js");
```

Volt syntax:

```
<?php
```

```
{{ javascript_include("http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/jquery.min.js", false) }}
{{ javascript_include("javascript/jquery.js") }}
```

public static *string* **image** ([*array* \$parameters], [*boolean* \$local])

Builds HTML IMG tags

```
<?php
```

```
echo Phalcon\Tag::image("img/bg.png");
echo Phalcon\Tag::image(array("img/photo.jpg", "alt" => "Some Photo"));
```

Volt Syntax:

```
<?php
```

```
{{ image("img/bg.png") }}
{{ image("img/photo.jpg", "alt": "Some Photo") }}
{{ image("http://static.mywebsite.com/img/bg.png", false) }}
```

public static *text* **friendlyTitle** (*string* \$text, [*string* \$separator], [*boolean* \$lowercase])

Converts texts into URL-friendly titles

```
<?php
```

```
echo Phalcon\Tag::friendlyTitle('These are big important news', '-')
```

public static **setDocType** (*string* \$doctype)

Set the document type of content

public static *string* **getDocType** ()

Get the document type declaration of content

public static *string* **tagHtml** (*string* \$tagName, [*array* \$parameters], [*boolean* \$selfClose], [*boolean* \$onlyStart], [*boolean* \$useEol])

Builds a HTML tag

```
<?php
```

```
echo Phalcon\Tag::tagHtml($name, $parameters, $selfClose, $onlyStart, $eol);
```

public static *string* **tagHtmlClose** (*string* \$tagName, [*boolean* \$useEol])

Builds a HTML tag closing tag

```
<?php
```

```
echo Phalcon\Tag::tagHtmlClose('script', true)
```

## 2.54.228 Class Phalcon\Tag\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Tag will use this class

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string* \$message], [*int* \$code], [*Exception* \$previous]) inherited from Exception

Exception constructor

final public *string* **getMessage** () inherited from Exception

Gets the Exception message

final public *int* **getCode** () inherited from Exception

Gets the Exception code

final public *string* **getFile** () inherited from Exception

Gets the file in which the exception occurred

final public *int* **getLine** () inherited from Exception

Gets the line in which the exception occurred

final public *array* **getTrace** () inherited from Exception

Gets the stack trace

final public *Exception* **getPrevious** () inherited from Exception

Returns previous Exception

final public *Exception* **getTraceAsString** () inherited from Exception

Gets the stack trace as a string

public *string* **\_\_toString** () inherited from Exception

String representation of the exception

## 2.54.229 Abstract class Phalcon\Tag\Select

Generates a SELECT html tag using a static array of values or a Phalcon\Mvc\Model resultset

### Methods

public static **selectField** (*array \$parameters, [array \$data]*)

Generates a SELECT tag

protected static **\_optionsFromResultset** ()

Generate the OPTION tags based on a resulset

protected static **\_optionsFromArray** ()

Generate the OPTION tags based on an array

## 2.54.230 Abstract class Phalcon\Text

Provides utilities to work with texts

### Constants

*integer RANDOM\_ALNUM*

*integer RANDOM\_ALPHA*

*integer RANDOM\_HEXDEC*

*integer RANDOM\_NUMERIC*

*integer RANDOM\_NOZERO*

### Methods

public static *string camelize (string \$str)*

Converts strings to camelize style

<?php

```
echo Phalcon\Text::camelize('coco_bongo'); //CocoBongo
```

public static *string uncamelize (string \$str)*

Uncamelize strings which are camelized

<?php

```
echo Phalcon\Text::uncamelize('CocoBongo'); //coco_bongo
```

public static *string increment (string \$str, [string \$separator])*

Adds a number to a string or increment that number if it already is defined

```
<?php
```

```
echo Phalcon\Text::increment("a"); // "a_1"
echo Phalcon\Text::increment("a_1"); // "a_2"
```

**public static string random (int \$type, [int \$length])**

Generates a random string based on the given type. Type is one of the RANDOM\_\* constants

```
<?php
```

```
echo Phalcon\Text::random(Phalcon\Text::RANDOM_ALNUM); // "aloiwkqz"
```

**public static boolean startsWith (string \$str, string \$start, [boolean \$ignoreCase])**

Check if a string starts with a given string

```
<?php
```

```
echo Phalcon\Text::startsWith("Hello", "He"); // true
echo Phalcon\Text::startsWith("Hello", "he"); // false
echo Phalcon\Text::startsWith("Hello", "he", false); // true
```

**public static boolean endsWith (string \$str, string \$end, [boolean \$ignoreCase])**

Check if a string ends with a given string

```
<?php
```

```
echo Phalcon\Text::endsWith("Hello", "llo"); // true
echo Phalcon\Text::endsWith("Hello", "LLO"); // false
echo Phalcon\Text::endsWith("Hello", "LLO", false); // true
```

**public static string lower (string \$str)**

Lowercases a string, this function makes use of the mbstring extension if available

**public static string upper (string \$str)**

Uppercases a string, this function makes use of the mbstring extension if available

## 2.54.231 Abstract class Phalcon\Translate\Adapter

*implements ArrayAccess, Phalcon\Translate\AdapterInterface*

Base class for Phalcon\Translate adapters

### Methods

**public \_\_construct ()**

Class constructore

**public string \_ (string \$translateKey, [array \$placeholders])**

Returns the translation string of the given key

**public offsetSet (unknown \$property, string \$value)**

Sets a translation value

**public boolean offsetExists (unknown \$property)**

Check whether a translation key exists

public **offsetUnset** (*unknown* \$property)

Unsets a translation from the dictionary

public *string* **offsetGet** (*unknown* \$property)

Returns the translation related to the given key

abstract public *string* **query** (*string* \$index, [*array* \$placeholders]) inherited from Phalcon\Translate\AdapterInterface

Returns the translation related to the given key

abstract public *bool* **exists** (*string* \$index) inherited from Phalcon\Translate\AdapterInterface

Check whether is defined a translation key in the internal array

## 2.54.232 Class Phalcon\Translate\Adapter\NativeArray

*extends* abstract class Phalcon\Translate\Adapter

*implements* Phalcon\Translate\AdapterInterface, ArrayAccess

Allows to define translation lists using PHP arrays

### Methods

public **\_\_construct** (*array* \$options)

Phalcon\Translate\Adapter\NativeArray constructor

public *string* **query** (*string* \$index, [*array* \$placeholders])

Returns the translation related to the given key

public *bool* **exists** (*string* \$index)

Check whether is defined a translation key in the internal array

public *string* **\_** (*string* \$translateKey, [*array* \$placeholders]) inherited from Phalcon\Translate\Adapter

Returns the translation string of the given key

public **offsetSet** (*unknown* \$property, *string* \$value) inherited from Phalcon\Translate\Adapter

Sets a translation value

public *boolean* **offsetExists** (*unknown* \$property) inherited from Phalcon\Translate\Adapter

Check whether a translation key exists

public **offsetUnset** (*unknown* \$property) inherited from Phalcon\Translate\Adapter

Unsets a translation from the dictionary

public *string* **offsetGet** (*unknown* \$property) inherited from Phalcon\Translate\Adapter

Returns the translation related to the given key

## 2.54.233 Class Phalcon\Translate\Exception

*extends class Phalcon\Exception*

Class for exceptions thrown by Phalcon\Translate

### Methods

final private *Exception* **\_\_clone** () inherited from Exception

Clone the exception

public **\_\_construct** ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

## 2.54.234 Class Phalcon\Validation

*extends abstract class Phalcon\DI\Injectable*

*implements Phalcon\Events\EventsAwareInterface, Phalcon\DI\InjectionAwareInterface*

Allows to validate data using validators

### Methods

public **\_\_construct** ([*array \$validators*])

Phalcon\Validation constructor

public *Phalcon\Validation\Message\Group validate* ([*array/object \$data*], [*object \$entity*])

Validate a set of data according to a set of rules

`public Phalcon\Validation add (string $attribute, unknown $validator)`

Adds a validator to a field

`public Phalcon\Validation setFilters (array/string $attribute, unknown $filters)`

Adds filters to the field

`public mixed getFilters ([string $attribute])`

Returns all the filters or a specific one

`public array getValidators ()`

Returns the validators added to the validation

`public object getEntity ()`

Returns the bound entity

`public Phalcon\Validation\Message\Group getMessages ()`

Returns the registered validators

`public Phalcon\Validation appendMessage (Phalcon\Validation\MessageInterface $message)`

Appends a message to the messages list

`public Phalcon\Validation bind (object $entity, object/array $data)`

Assigns the data to an entity The entity is used to obtain the validation values

`public mixed getValue (string $attribute)`

Gets the a value to validate in the array/object data source

`public setDefaultMessages ([unknown $messages])`

...

`public getDefaultMessage (unknown $type)`

...

`public setLabels (unknown $labels)`

Adds labels for fields

`public mixed getLabel (unknown $field)`

Get label for field

`public setDI (Phalcon\DiInterface $dependencyInjector)` inherited from Phalcon\DI\Injectable

Sets the dependency injector

`public Phalcon\DiInterface getDI ()` inherited from Phalcon\DI\Injectable

Returns the internal dependency injector

`public setEventsManager (Phalcon\Events\ManagerInterface $eventsManager)` inherited from Phalcon\DI\Injectable

Sets the event manager

`public Phalcon\Events\ManagerInterface getEventsManager ()` inherited from Phalcon\DI\Injectable

Returns the internal event manager

public `__get` (*unknown \$property*) inherited from Phalcon\DI\Injectable  
Magic method `__get`

### 2.54.235 Class Phalcon\Validation\Exception

*extends class Phalcon\Exception*

Exceptions thrown in Phalcon\Validation\\* classes will use this class

#### Methods

final private *Exception* `__clone` () inherited from Exception

Clone the exception

public `__construct` ([*string \$message*], [*int \$code*], [*Exception \$previous*]) inherited from Exception

Exception constructor

final public *string getMessage* () inherited from Exception

Gets the Exception message

final public *int getCode* () inherited from Exception

Gets the Exception code

final public *string getFile* () inherited from Exception

Gets the file in which the exception occurred

final public *int getLine* () inherited from Exception

Gets the line in which the exception occurred

final public *array getTrace* () inherited from Exception

Gets the stack trace

final public *Exception getPrevious* () inherited from Exception

Returns previous Exception

final public *Exception getTraceAsString* () inherited from Exception

Gets the stack trace as a string

public *string \_\_toString* () inherited from Exception

String representation of the exception

### 2.54.236 Class Phalcon\Validation\Message

Encapsulates validation info generated in the validation process

## Methods

```
public __construct (string $message, [string $field], [string $type], [int $code])  
Phalcon\Validation\Message constructor  
public Phalcon\Validation\Message setType (string $type)  
Sets message type  
public string getType ()  
Returns message type  
public Phalcon\Validation\Message setCode (string $code)  
Sets message code  
public string getCode ()  
Returns message code  
public Phalcon\Validation\Message setMessage (string $message)  
Sets verbose message  
public string getMessage ()  
Returns verbose message  
public Phalcon\Validation\Message setField (string $field)  
Sets field name related to message  
public string getField ()  
Returns field name related to message  
public string __toString ()  
Magic __toString method returns verbose message  
public static Phalcon\Validation\Message __set_state (array $message)  
Magic __set_state helps to recover messages from serialization
```

## 2.54.237 Class Phalcon\Validation\Message\Group

*implements* Countable, ArrayAccess, Iterator, Traversable

Represents a group of validation messages

## Methods

```
public __construct ([array $messages])  
Phalcon\Validation\Message\Group constructor  
public Phalcon\Validation\Message offsetGet (string $index)  
Gets an attribute a message using the array syntax  
<?php  
print_r($messages[0]);
```

**public offsetSet (string \$index, Phalcon\Validation|Message \$message)**

Sets an attribute using the array-syntax

<?php

```
$messages[0] = new Phalcon\Validation\Message('This is a message');
```

**public boolean offsetExists (string \$index)**

Checks if an index exists

<?php

```
var_dump(isset($message['database']));
```

**public offsetUnset (string \$index)**

Removes a message from the list

<?php

```
unset($message['database']);
```

**public appendMessage (Phalcon\Validation|Message \$message)**

Appends a message to the group

<?php

```
$messages->appendMessage(new Phalcon\Validation\Message('This is a message'));
```

**public appendMessages (Phalcon\Validation\MessageInterface[] \$messages)**

Appends an array of messages to the group

<?php

```
$messages->appendMessages($messagesArray);
```

**public array filter (string \$fieldName)**

Filters the message group by field name

**public int count ()**

Returns the number of messages in the list

**public rewind ()**

Rewinds the internal iterator

**public Phalcon\Validation\Message current ()**

Returns the current message in the iterator

**public int key ()**

Returns the current position/key in the iterator

**public next ()**

Moves the internal iteration pointer to the next position

**public boolean valid ()**

Check if the current message in the iterator is valid

public static *Phalcon\ Mvc\ Model\ Message\ Group* **\_\_set\_state** (*array \$group*)

Magic **\_\_set\_state** helps to re-build messages variable when exporting

## 2.54.238 Abstract class Phalcon\Validation\Validator

*implements Phalcon\Validation\ValidatorInterface*

This is a base class for validators

### Methods

public **\_\_construct** ([*array \$options*])

Phalcon\Validation\Validator constructor

public *mixed isSetOption* (*string \$key*)

Checks if an option is defined

public *mixed getOption* (*string \$key*)

Returns an option in the validator's options Returns null if the option hasn't been set

public **setOption** (*string \$key, mixed \$value*)

Sets an option in the validator

abstract public *Phalcon\Validation\Message\Group validate* (*Phalcon\Validator \$validator, string \$attribute*) inherited from Phalcon\Validation\ValidatorInterface

Executes the validation

## 2.54.239 Class Phalcon\Validation\Validator\Between

*extends abstract class Phalcon\Validation\Validator*

*implements Phalcon\Validation\ValidatorInterface*

Validates that a value is between a range of two values

<?php

```
use Phalcon\Validation\Validator\Between;

$validator->add('name', new Between(array(
    'minimum' => 0,
    'maximum' => 100,
    'message' => 'The price must be between 0 and 100',
)));



```

### Methods

public *boolean validate* (*Phalcon\Validation \$validator, string \$attribute*)

Executes the validation

public **\_\_construct** ([*array \$options*]) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed* **isSetOption** (*string* \$key) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed* **getOption** (*string* \$key) inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public **setOption** (*string* \$key, *mixed* \$value) inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.240 Class Phalcon\Validation\Validator\Confirmation

*extends* abstract class Phalcon\Validation\Validator

*implements* Phalcon\Validation\ValidatorInterface

Checks that two values have the same value

<?php

```
use Phalcon\Validation\Validator\Confirmation;

$validator->add('password', new Confirmation(array(
    'message' => 'Password doesn\'t match confirmation',
    'with' => 'confirmPassword'
))');
```

### Methods

public *boolean* **validate** (*Phalcon\Validation* \$validator, *string* \$attribute)

Executes the validation

public **\_\_construct** ([*array* \$options]) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed* **isSetOption** (*string* \$key) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed* **getOption** (*string* \$key) inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public **setOption** (*string* \$key, *mixed* \$value) inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.241 Class Phalcon\Validation\Validator\Email

*extends* abstract class Phalcon\Validation\Validator

*implements* Phalcon\Validation\ValidatorInterface

Checks if a value has a correct e-mail format

```
<?php

use Phalcon\Validation\Validator\Email as EmailValidator;

$validator->add('email', new EmailValidator(array(
    'message' => 'The e-mail is not valid'
)));



```

## Methods

public *boolean validate* (*Phalcon|Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct* (*[array \$options]*) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed getOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public *setOption* (*string \$key, mixed \$value*) inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.242 Class Phalcon\Validation\Validator\ExclusionIn

*extends abstract class Phalcon\Validation\Validator*

*implements Phalcon\Validation\ValidatorInterface*

Check if a value is not included into a list of values

```
<?php
```

```
use Phalcon\Validation\Validator\ExclusionIn;
```

```
$validator->add('status', new ExclusionIn(array(
```

```
    'message' => 'The status must not be A or B',
```

```
    'domain' => array('A', 'B')
```

```
));
```

## Methods

public *boolean validate* (*Phalcon|Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct* (*[array \$options]*) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed* **getOption** (*string* \$key) inherited from Phalcon\Validation\Validator  
 Returns an option in the validator's options Returns null if the option hasn't been set

public **setOption** (*string* \$key, *mixed* \$value) inherited from Phalcon\Validation\Validator  
 Sets an option in the validator

### 2.54.243 Class Phalcon\Validation\Validator\Identical

*extends* abstract class Phalcon\Validation\Validator

*implements* Phalcon\Validation\ValidatorInterface

Checks if a value is identical to other

```
<?php
```

```
use Phalcon\Validation\Validator\Identical;

$validator->add('terms', new Identical(array(
    'value' => 'yes',
    'message' => 'Terms and conditions must be accepted'
)));"
```

#### Methods

public *boolean* **validate** (Phalcon\Validation \$validator, *string* \$attribute)

Executes the validation

public **\_\_construct** ([*array* \$options]) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed* **isSetOption** (*string* \$key) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed* **getOption** (*string* \$key) inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public **setOption** (*string* \$key, *mixed* \$value) inherited from Phalcon\Validation\Validator

Sets an option in the validator

### 2.54.244 Class Phalcon\Validation\Validator\InclusionIn

*extends* abstract class Phalcon\Validation\Validator

*implements* Phalcon\Validation\ValidatorInterface

Check if a value is included into a list of values

```
<?php
```

```
use Phalcon\Validation\Validator\InclusionIn;
```

```
$validator->add('status', new InclusionIn(array(
    'message' => 'The status must be A or B',
```

```
'domain' => array('A', 'B')
));;
```

## Methods

public *boolean validate* (*Phalcon|Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct ([array \$options])* inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption (string \$key)* inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed getOption (string \$key)* inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public *setOption (string \$key, mixed \$value)* inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.245 Class Phalcon\Validation\Validator\PresenceOf

*extends abstract class Phalcon\Validation\Validator*

*implements Phalcon\Validation\ValidatorInterface*

Validates that a value is not null or empty string

```
<?php
```

```
use Phalcon\Validation\Validator\PresenceOf;
```

```
$validator->add('name', new PresenceOf(array(
    'message' => 'The name is required',
)));
```

## Methods

public *boolean validate* (*Phalcon|Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct ([array \$options])* inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption (string \$key)* inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed getOption (string \$key)* inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public *setOption (string \$key, mixed \$value)* inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.246 Class Phalcon\Validation\Validator\Regex

*extends abstract class Phalcon\Validation\Validator*  
*implements Phalcon\Validation\ValidatorInterface*

Allows validate if the value of a field matches a regular expression

```
<?php

use Phalcon\Validation\Validator\Regex as RegexValidator;

$validator->add('created_at', new RegexValidator(array(
    'pattern' => '/^([0-9]{4})[-/](0[1-9]|1[0-2])[-/](0[1-9]|1[0-2])[-/](0[1-9]|3[01])$/',
    'message' => 'The creation date is invalid',
)));



```

### Methods

public *boolean validate* (*Phalcon\Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct ([array \$options])* inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption (string \$key)* inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed getOption (string \$key)* inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public *setOption (string \$key, mixed \$value)* inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.247 Class Phalcon\Validation\Validator\StringLength

*extends abstract class Phalcon\Validation\Validator*

*implements Phalcon\Validation\ValidatorInterface*

Validates that a string has the specified maximum and minimum constraints

```
<?php



```

```
use Phalcon\Validation\Validator\StringLength as StringLength;
```

```
$validation->add('name_last', new StringLength(array(
    'max' => 50,
    'min' => 2,
    'messageMaximum' => 'We don\'t like really long names',
    'messageMinimum' => 'We want more than just their initials',
)));



```

## Methods

public *boolean validate* (*Phalcon|Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct* (*[array \$options]*) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed getOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public *setOption* (*string \$key, mixed \$value*) inherited from Phalcon\Validation\Validator

Sets an option in the validator

### 2.54.248 Class Phalcon\Validation\Validator\Url

*extends abstract class Phalcon\Validation\Validator*

*implements Phalcon\Validation\ValidatorInterface*

Checks if a value has a correct URL format

<?php

```
use Phalcon\Validation\Validator\Url as UrlValidator;

$validator->add('url', new UrlValidator(array(
    'message' => 'The url is not valid'
)));



```

## Methods

public *boolean validate* (*Phalcon|Validation \$validator, string \$attribute*)

Executes the validation

public *\_\_construct* (*[array \$options]*) inherited from Phalcon\Validation\Validator

Phalcon\Validation\Validator constructor

public *mixed isSetOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Checks if an option is defined

public *mixed getOption* (*string \$key*) inherited from Phalcon\Validation\Validator

Returns an option in the validator's options Returns null if the option hasn't been set

public *setOption* (*string \$key, mixed \$value*) inherited from Phalcon\Validation\Validator

Sets an option in the validator

## 2.54.249 Class Phalcon\Version

This class allows to get the installed version of the framework

### Methods

protected static **\_getVersion ()**

Area where the version number is set. The format is as follows: ABCDE A - Major version B - Med version (two digits) C - Min version (two digits) D - Special release: 1 = Alpha, 2 = Beta, 3 = RC, 4 = Stable E - Special release version i.e. RC1, Beta2 etc.

public static *string* **get ()**

Returns the active version (*string*)

<?php

```
echo Phalcon\Version::get();
```

public static *int* **getId ()**

Returns the numeric active version

<?php

```
echo Phalcon\Version::getId();
```

## 2.54.250 Interface Phalcon\Acl\AdapterInterface

Phalcon\Acl\AdapterInterface initializer

### Methods

abstract public **setDefaultAction (int \$defaultAccess)**

Sets the default access level (Phalcon\Acl::ALLOW or Phalcon\Acl::DENY)

abstract public *int* **getDefaultAction ()**

Returns the default ACL access level

abstract public *boolean* **addRole (*Phalcon\Acl\RoleInterface* \$role, [*string* \$accessInherits])**

Adds a role to the ACL list. Second parameter lets to inherit access data from other existing role

abstract public **addInherit (*string* \$roleName, *string* \$roleToInherit)**

Do a role inherit from another existing role

abstract public *boolean* **isRole (*string* \$roleName)**

Check whether role exist in the roles list

abstract public *boolean* **isResource (*string* \$resourceName)**

Check whether resource exist in the resources list

abstract public *boolean* **addResource (*Phalcon\Acl\ResourceInterface* \$resource, [*array* \$accessList])**

Adds a resource to the ACL list Access names can be a particular action, by example search, update, delete, etc or a list of them

abstract public **addResourceAccess** (*string \$resourceName, mixed \$accessList*)

Adds access to resources

abstract public **dropResourceAccess** (*string \$resourceName, mixed \$accessList*)

Removes an access from a resource

abstract public **allow** (*string \$roleName, string \$resourceName, mixed \$access*)

Allow access to a role on a resource

abstract public *boolean deny* (*string \$roleName, string \$resourceName, mixed \$access*)

Deny access to a role on a resource

abstract public *boolean isAllowed* (*string \$role, string \$resource, string \$access*)

Check whether a role is allowed to access an action from a resource

abstract public *string getActiveRole* ()

Returns the role which the list is checking if it's allowed to certain resource/access

abstract public *string getActiveResource* ()

Returns the resource which the list is checking if some role can access it

abstract public *string getActiveAccess* ()

Returns the access which the list is checking if some role can access it

abstract public *Phalcon\Acl\RoleInterface [] getRoles* ()

Return an array with every role registered in the list

abstract public *Phalcon\Acl\ResourceInterface [] getResources* ()

Return an array with every resource registered in the list

## 2.54.251 Interface Phalcon\Acl\ResourceInterface

Phalcon\Acl\ResourceInterface initializer

### Methods

abstract public *string getName* ()

Returns the resource name

abstract public *string getDescription* ()

Returns resource description

## 2.54.252 Interface Phalcon\Acl\RoleInterface

Phalcon\Acl\RoleInterface initializer

## Methods

abstract public *string* **getName** ()

Returns the role name

abstract public *string* **getDescription** ()

Returns role description

## 2.54.253 Interface Phalcon\Annotations\AdapterInterface

Phalcon\Annotations\AdapterInterface initializer

## Methods

abstract public *Phalcon\Annotations\Reflection* **read** (*string* \$key)

Read parsed annotations

abstract public **write** (*string* \$key, *Phalcon\Annotations\Reflection* \$data)

Write parsed annotations

abstract public **setReader** (*Phalcon\Annotations\ReaderInterface* \$reader)

Sets the annotations parser

abstract public *Phalcon\Annotations\ReaderInterface* **getReader** ()

Returns the annotation reader

abstract public *Phalcon\Annotations\Reflection* **get** (*string/object* \$className)

Parses or retrieves all the annotations found in a class

abstract public *array* **getMethods** (*string* \$className)

Returns the annotations found in all the class' methods

abstract public *Phalcon\Annotations\Collection* **getMethod** (*string* \$className, *string* \$methodName)

Returns the annotations found in a specific method

abstract public *array* **getProperties** (*string* \$className)

Returns the annotations found in all the class' properties

abstract public *Phalcon\Annotations\Collection* **getProperty** (*string* \$className, *string* \$propertyName)

Returns the annotations found in a specific property

## 2.54.254 Interface Phalcon\Annotations\ReaderInterface

Phalcon\Annotations\ReaderInterface initializer

## Methods

abstract public **array parse** (*string \$className*)

Reads annotations from the class dockblocks, its methods and/or properties

abstract public static **array parseDocBlock** (*string \$docBlock, [unknown \$file], [unknown \$line]*)

Parses a raw doc block returning the annotations found

## 2.54.255 Interface Phalcon\Assets\FilterInterface

Phalcon\Assets\FilterInterface initializer

## Methods

abstract public **\$content filter** (*string \$content*)

Filters the content returning a string with the filtered content

## 2.54.256 Interface Phalcon\Cache\BackendInterface

Phalcon\Cache\BackendInterface initializer

## Methods

abstract public **mixed start** (*int/string \$keyName, [long \$lifetime]*)

Starts a cache. The \$keyname allows to identify the created fragment

abstract public **stop** (*[boolean \$stopBuffer]*)

Stops the frontend without store any cached content

abstract public **mixed getFrontend** ()

Returns front-end instance adapter related to the back-end

abstract public **array getOptions** ()

Returns the backend options

abstract public **boolean isFresh** ()

Checks whether the last cache is fresh or cached

abstract public **boolean isStarted** ()

Checks whether the cache has starting buffering or not

abstract public **setLastKey** (*string \$lastKey*)

Sets the last key used in the cache

abstract public **string getLastKey** ()

Gets the last key stored by the cache

abstract public **mixed get** (*int/string \$keyName, [long \$lifetime]*)

Returns a cached content

abstract public **save** ([*int/string* \$keyName], [*string* \$content], [*long* \$lifetime], [*boolean* \$stopBuffer])

Stores cached content into the file backend and stops the frontend

abstract public *boolean* **delete** (*int/string* \$keyName)

Deletes a value from the cache by its key

abstract public *array* **queryKeys** ([*string* \$prefix])

Query the existing cached keys

abstract public *boolean* **exists** ([*string* \$keyName], [*long* \$lifetime])

Checks if cache exists and it hasn't expired

abstract public *boolean* **flush** ()

Immediately invalidates all existing items.

## 2.54.257 Interface Phalcon\Cache\FrontendInterface

Phalcon\Cache\FrontendInterface initializer

### Methods

abstract public *int* **getLifetime** ()

Returns the cache lifetime

abstract public *boolean* **isBuffering** ()

Check whether if frontend is buffering output

abstract public **start** ()

Starts the frontend

abstract public *string* **getContent** ()

Returns output cached content

abstract public **stop** ()

Stops the frontend

abstract public **beforeStore** (*mixed* \$data)

Serializes data before storing it

abstract public **afterRetrieve** (*mixed* \$data)

Unserializes data after retrieving it

## 2.54.258 Interface Phalcon\CryptInterface

Phalcon\CryptInterface initializer

## Methods

abstract public *Phalcon\|CryptInterface* **setCipher** (*string* \$cipher)

Sets the cipher algorithm

abstract public *string* **getCipher** ()

Returns the current cipher

abstract public *Phalcon\|CryptInterface* **setMode** (*unknown* \$mode)

Sets the encrypt/decrypt mode

abstract public *string* **getMode** ()

Returns the current encryption mode

abstract public *Phalcon\|CryptInterface* **setKey** (*string* \$key)

Sets the encryption key

abstract public *string* **getKey** ()

Returns the encryption key

abstract public *string* **encrypt** (*string* \$text, [*string* \$key])

Encrypts a text

abstract public *string* **decrypt** (*string* \$text, [*string* \$key])

Decrypts a text

abstract public *string* **encryptBase64** (*string* \$text, [*string* \$key], [*unknown* \$safe])

Encrypts a text returning the result as a base64 string

abstract public *string* **decryptBase64** (*string* \$text, [*string* \$key], [*unknown* \$safe])

Decrypt a text that is coded as a base64 string

abstract public *array* **getAvailableCiphers** ()

Returns a list of available cyphers

abstract public *array* **getAvailableModes** ()

Returns a list of available modes

## 2.54.259 Interface Phalcon\DI\InjectionAwareInterface

Phalcon\DI\InjectionAwareInterface initializer

## Methods

abstract public **setDI** (*Phalcon\|DiInterface* \$dependencyInjector)

Sets the dependency injector

abstract public *Phalcon\|DiInterface* **getDI** ()

Returns the internal dependency injector

## 2.54.260 Interface Phalcon\DI\ServiceInterface

Phalcon\DI\ServiceInterface initializer

### Methods

abstract public *string* **getName** ()

Returns the name of the service

abstract public **setShared** (*boolean* \$shared)

Sets whether the service is shared or not

abstract public *boolean* **isShared** ()

Check whether the service is shared or not

abstract public **setDefinition** (*mixed* \$definition)

Set the service definition

abstract public *mixed* **getDefinition** ()

Returns the service definition

abstract public *bool* **isResolved** ()

Checks if the service was resolved

abstract public *object* **resolve** ([*array* \$parameters], [*Phalcon\DiInterface* \$dependencyInjector])

Resolves the service

## 2.54.261 Interface Phalcon\Db\AdapterInterface

Phalcon\Db\AdapterInterface initializer

### Methods

abstract public *array* **fetchOne** (*string* \$sqlQuery, [*int* \$fetchMode], [*int* \$placeholders])

Returns the first row in a SQL query result

abstract public *array* **fetchAll** (*string* \$sqlQuery, [*int* \$fetchMode], [*int* \$placeholders])

Dumps the complete result of a query into an array

abstract public *boolean* **insert** (*string* \$table, *array* \$values, [*array* \$fields], [*array* \$dataTypes])

Inserts data into a table using custom RBDM SQL syntax

abstract public *boolean* **update** (*string* \$table, *array* \$fields, *array* \$values, [*string* \$whereCondition], [*array* \$dataTypes])

Updates data on a table using custom RBDM SQL syntax

abstract public *boolean* **delete** (*string* \$table, [*string* \$whereCondition], [*array* \$placeholders], [*array* \$dataTypes])

Deletes data from a table using custom RBDM SQL syntax

abstract public *string* **getColumnList** (*array* \$columnList)

Gets a list of columns

```
abstract public string limit (string $sqlQuery, int $number)
```

Appends a LIMIT clause to \$sqlQuery argument

```
abstract public string tableExists (string $tableName, [string $schemaName])
```

Generates SQL checking for the existence of a schema.table

```
abstract public string viewExists (string $viewName, [string $schemaName])
```

Generates SQL checking for the existence of a schema.view

```
abstract public string forUpdate (string $sqlQuery)
```

Returns a SQL modified with a FOR UPDATE clause

```
abstract public string sharedLock (string $sqlQuery)
```

Returns a SQL modified with a LOCK IN SHARE MODE clause

```
abstract public boolean createTable (string $tableName, string $schemaName, array $definition)
```

Creates a table

```
abstract public boolean dropTable (string $tableName, [string $schemaName], [boolean $IfExists])
```

Drops a table from a schema/database

```
abstract public boolean createView (unknown $viewName, array $definition, [string $schemaName])
```

Creates a view

```
abstract public boolean dropView (string $viewName, [string $schemaName], [boolean $IfExists])
```

Drops a view

```
abstract public boolean addColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)
```

Adds a column to a table

```
abstract public boolean modifyColumn (string $tableName, string $schemaName, Phalcon\Db\ColumnInterface $column)
```

Modifies a table column based on a definition

```
abstract public boolean dropColumn (string $tableName, string $schemaName, string $columnName)
```

Drops a column from a table

```
abstract public boolean addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)
```

Adds an index to a table

```
abstract public boolean dropIndex (string $tableName, string $schemaName, string $indexName)
```

Drop an index from a table

```
abstract public boolean addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)
```

Adds a primary key to a table

```
abstract public boolean dropPrimaryKey (string $tableName, string $schemaName)
```

Drops primary key from a table

---

```
abstract public boolean true addForeignKey (string $tableName, string $schemaName,
Phalcon\Db\ReferenceInterface $reference)
```

Adds a foreign key to a table

```
abstract public boolean true dropForeignKey (string $tableName, string $schemaName, string
$referenceName)
```

Drops a foreign key from a table

```
abstract public string getColumnDefinition (Phalcon\Db\ColumnInterface $column)
```

Returns the SQL column definition from a column

```
abstract public array listTables ([string $schemaName])
```

List all tables on a database

```
abstract public array listViews ([string $schemaName])
```

List all views on a database

```
abstract public array getDescriptor ()
```

Return descriptor used to connect to the active database

```
abstract public string getConnectionId ()
```

Gets the active connection unique identifier

```
abstract public string getSQLStatement ()
```

Active SQL statement in the object

```
abstract public string getRealSQLStatement ()
```

Active SQL statement in the object without replace bound paramters

```
abstract public array getSQLVariables ()
```

Active SQL statement in the object

```
abstract public array getSQLBindTypes ()
```

Active SQL statement in the object

```
abstract public string getType ()
```

Returns type of database system the adapter is used for

```
abstract public string getDialectType ()
```

Returns the name of the dialect used

```
abstract public Phalcon\Db\DialectInterface getDialect ()
```

Returns internal dialect instance

```
abstract public boolean connect ([array $descriptor])
```

This method is automatically called in Phalcon\Db\Adapter\Pdo constructor. Call it when you need to restore a database connection

```
abstract public Phalcon\Db\ResultInterface query (string $sqlStatement, [array $placeholders], [array
$dataTypes])
```

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server return rows

```
abstract public boolean execute (string $sqlStatement, [array $placeholders], [array $dataTypes])
```

Sends SQL statements to the database server returning the success state. Use this method only when the SQL statement sent to the server don't return any row

abstract public *int* **affectedRows** ()

Returns the number of affected rows by the last INSERT/UPDATE/DELETE reported by the database system

abstract public *boolean* **close** ()

Closes active connection returning success. Phalcon automatically closes and destroys active connections within Phalcon\Db\Pool

abstract public *string* **escapeIdentifier** (*string* \$identifier)

Escapes a column/table/schema name

abstract public *string* **escapeString** (*string* \$str)

Escapes a value to avoid SQL injections

abstract public *array* **convertBoundParams** (*string* \$sqlStatement, *array* \$params)

Converts bound params like :name: or ?1 into ? bind params

abstract public *int* **lastInsertId** ([*string* \$sequenceName])

Returns insert id for the auto\_increment column inserted in the last SQL statement

abstract public *boolean* **begin** ()

Starts a transaction in the connection

abstract public *boolean* **rollback** ()

Rollbacks the active transaction in the connection

abstract public *boolean* **commit** ()

Commits the active transaction in the connection

abstract public *boolean* **isUnderTransaction** ()

Checks whether connection is under database transaction

abstract public *PDO* **getInternalHandler** ()

Return internal PDO handler

abstract public *Phalcon\Db\IndexInterface* [] **describeIndexes** (*string* \$table, [*string* \$schema])

Lists table indexes

abstract public *Phalcon\Db\ReferenceInterface* [] **describeReferences** (*string* \$table, [*string* \$schema])

Lists table references

abstract public *array* **tableOptions** (*string* \$tableName, [*string* \$schemaName])

Gets creation options from a table

abstract public *boolean* **useExplicitIdValue** ()

Check whether the database system requires an explicit value for identity columns

abstract public *Phalcon\Db\RawValue* **getDefaultIdValue** ()

Return the default identity value to insert in an identity column

abstract public *boolean* **supportSequences** ()

Check whether the database system requires a sequence to produce auto-numeric values

`abstract public boolean createSavepoint (string $name)`

Creates a new savepoint

`abstract public boolean releaseSavepoint (string $name)`

Releases given savepoint

`abstract public boolean rollbackSavepoint (string $name)`

Rollbacks given savepoint

`abstract public Phalcon\Db\AdapterInterface setNestedTransactionsWithSavepoints (boolean $nestedTransactionsWithSavepoints)`

Set if nested transactions should use savepoints

`abstract public boolean isNestedTransactionsWithSavepoints ()`

Returns if nested transactions should use savepoints

`abstract public string getNestedTransactionSavepointName ()`

Returns the savepoint name to use for nested transactions

`abstract public Phalcon\Db\ColumnInterface [] describeColumns (string $table, [string $schema])`

Returns an array of Phalcon\Db\Column objects describing a table

## 2.54.262 Interface Phalcon\Db\ColumnInterface

Phalcon\Db\ColumnInterface initializer

### Methods

`abstract public string getSchemaName ()`

Returns schema's table related to column

`abstract public string getName ()`

Returns column name

`abstract public int getType ()`

Returns column type

`abstract public int getSize ()`

Returns column size

`abstract public int getScale ()`

Returns column scale

`abstract public boolean isUnsigned ()`

Returns true if number column is unsigned

`abstract public boolean isNotNull ()`

Not null

`abstract public boolean isPrimary ()`

Column is part of the primary key?

abstract public *boolean* **isAutoIncrement** ()

Auto-Increment

abstract public *boolean* **isNumeric** ()

Check whether column have an numeric type

abstract public *boolean* **isFirst** ()

Check whether column have first position in table

abstract public *string* **getAfterPosition** ()

Check whether field absolute to position in table

abstract public *int* **getBindType** ()

Returns the type of bind handling

## 2.54.263 Interface Phalcon\Db\DialectInterface

Phalcon\Db\DialectInterface initializer

### Methods

abstract public *string* **limit** (*string* \$sqlQuery, *int* \$number)

Generates the SQL for LIMIT clause

abstract public *string* **forUpdate** (*string* \$sqlQuery)

Returns a SQL modified with a FOR UPDATE clause

abstract public *string* **sharedLock** (*string* \$sqlQuery)

Returns a SQL modified with a LOCK IN SHARE MODE clause

abstract public *string* **select** (*array* \$definition)

Builds a SELECT statement

abstract public *string* **getColumnList** (*array* \$columnList)

Gets a list of columns

abstract public *string* **getColumnDefinition** (*Phalcon\Db\ColumnInterface* \$column)

Gets the column name in MySQL

abstract public *string* **addColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column)

Generates SQL to add a column to a table

abstract public *string* **modifyColumn** (*string* \$tableName, *string* \$schemaName, *Phalcon\Db\ColumnInterface* \$column)

Generates SQL to modify a column in a table

abstract public *string* **dropColumn** (*string* \$tableName, *string* \$schemaName, *string* \$columnName)

Generates SQL to delete a column from a table

```
abstract public string addIndex (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)
```

Generates SQL to add an index to a table

```
abstract public string dropIndex (string $tableName, string $schemaName, string $indexName)
```

Generates SQL to delete an index from a table

```
abstract public string addPrimaryKey (string $tableName, string $schemaName, Phalcon\Db\IndexInterface $index)
```

Generates SQL to add the primary key to a table

```
abstract public string dropPrimaryKey (string $tableName, string $schemaName)
```

Generates SQL to delete primary key from a table

```
abstract public string addForeignKey (string $tableName, string $schemaName, Phalcon\Db\ReferenceInterface $reference)
```

Generates SQL to add an index to a table

```
abstract public string dropForeignKey (string $tableName, string $schemaName, string $referenceName)
```

Generates SQL to delete a foreign key from a table

```
abstract public string createTable (string $tableName, string $schemaName, array $definition)
```

Generates SQL to create a table

```
abstract public string dropTable (string $tableName, string $schemaName)
```

Generates SQL to drop a table

```
abstract public string createView (string $viewName, array $definition, string $schemaName)
```

Generates SQL to create a view

```
abstract public string dropView (string $viewName, string $schemaName, [unknown $IfExists])
```

Generates SQL to drop a view

```
abstract public string tableExists (string $tableName, [string $schemaName])
```

Generates SQL checking for the existence of a schema.table

```
abstract public string viewExists (string $viewName, [string $schemaName])
```

Generates SQL checking for the existence of a schema.view

```
abstract public string describeColumns (string $table, [string $schema])
```

Generates SQL to describe a table

```
abstract public array listTables ([string $schemaName])
```

List all tables on database

```
abstract public array listViews ([string $schemaName])
```

List all views on database

```
abstract public string describeIndexes (string $table, [string $schema])
```

Generates SQL to query indexes on a table

```
abstract public string describeReferences (string $table, [string $schema])
```

Generates SQL to query foreign keys on a table

abstract public *string* **tableOptions** (*string* \$table, [*string* \$schema])

Generates the SQL to describe the table creation options

abstract public *boolean* **supportsSavepoints** ()

Checks whether the platform supports savepoints

abstract public *boolean* **supportsReleaseSavepoints** ()

Checks whether the platform supports releasing savepoints.

abstract public *string* **createSavepoint** (*string* \$name)

Generate SQL to create a new savepoint

abstract public *string* **releaseSavepoint** (*string* \$name)

Generate SQL to release a savepoint

abstract public *string* **rollbackSavepoint** (*string* \$name)

Generate SQL to rollback a savepoint

## 2.54.264 Interface Phalcon\Db\IndexInterface

Phalcon\Db\IndexInterface initializer

### Methods

abstract public *string* **getName** ()

Gets the index name

abstract public *array* **getColumns** ()

Gets the columns that comprehends the index

## 2.54.265 Interface Phalcon\Db\ReferenceInterface

Phalcon\Db\ReferenceInterface initializer

### Methods

abstract public *string* **getName** ()

Gets the index name

abstract public *string* **getSchemaName** ()

Gets the schema where referenced table is

abstract public *string* **getReferencedSchema** ()

Gets the schema where referenced table is

abstract public *array* **getColumns** ()

Gets local columns which reference is based

abstract public *string* **getReferencedTable** ()

Gets the referenced table

**abstract public array getReferencedColumns ()**

Gets referenced columns

## 2.54.266 Interface Phalcon\Db\ResultInterface

Phalcon\Db\ResultInterface initializer

### Methods

**abstract public boolean execute ()**

Allows to executes the statement again. Some database systems don't support scrollable cursors, So, as cursors are forward only, we need to execute the cursor again to fetch rows from the begining

**abstract public mixed fetch ()**

Fetches an array/object of strings that corresponds to the fetched row, or FALSE if there are no more rows. This method is affected by the active fetch flag set using Phalcon\Db\Result\Pdo::setFetchMode

**abstract public mixed fetchArray ()**

Returns an array of strings that corresponds to the fetched row, or FALSE if there are no more rows. This method is affected by the active fetch flag set using Phalcon\Db\Result\Pdo::setFetchMode

**abstract public array fetchAll ()**

Returns an array of arrays containing all the records in the result This method is affected by the active fetch flag set using Phalcon\Db\Result\Pdo::setFetchMode

**abstract public int numRows ()**

Gets number of rows returned by a resulset

**abstract public dataSeek (int \$number)**

Moves internal resulset cursor to another position letting us to fetch a certain row

**abstract public setFetchMode (int \$fetchMode, [unknown \$fetchArg1], [unknown \$fetchArg2])**

Changes the fetching mode affecting Phalcon\Db\Result\Pdo::fetch()

**abstract public PDOStatement getInternalResult ()**

Gets the internal PDO result object

## 2.54.267 Interface Phalcon\DiInterface

*extends* ArrayAccess

Phalcon\DiInterface initializer

### Methods

**abstract public Phalcon\DI\ServiceInterface set (string \$name, mixed \$definition, [boolean \$shared])**

Registers a service in the service container

**abstract public remove (string \$name)**

Removes a service from the service container

abstract public *object* **get** (*string* \$name, [*array* \$parameters])

Resolves the service based on its configuration

abstract public *object* **getShared** (*string* \$name, [*array* \$parameters])

Resolves a shared service based on their configuration

abstract public *Phalcon\DI\ServiceInterface* **setService** (*Phalcon\DI\ServiceInterface* \$rawDefinition)

Sets a service using a raw Phalcon\DI\Service definition

abstract public *Phalcon\DI\ServiceInterface* **getService** (*string* \$name)

Returns the corresponding Phalcon\Di\Service instance for a service

abstract public *boolean* **has** (*string* \$name)

Check whether the DI contains a service by a name

abstract public *boolean* **wasFreshInstance** ()

Check whether the last service obtained via getShared produced a fresh instance or an existing one

abstract public *array* **getServices** ()

Return the services registered in the DI

abstract public static **setDefault** (*Phalcon\DiInterface* \$dependencyInjector)

Set the default dependency injection container to be obtained into static methods

abstract public static *Phalcon\DiInterface* **getDefault** ()

Return the last DI created

abstract public static **reset** ()

Resets the internal default DI

abstract public **offsetExists** (*unknown* \$offset) inherited from ArrayAccess

...

abstract public **offsetGet** (*unknown* \$offset) inherited from ArrayAccess

...

abstract public **offsetSet** (*unknown* \$offset, *unknown* \$value) inherited from ArrayAccess

...

abstract public **offsetUnset** (*unknown* \$offset) inherited from ArrayAccess

...

## 2.54.268 Interface Phalcon\DispatcherInterface

Phalcon\DispatcherInterface initializer

## Methods

abstract public **setActionSuffix** (*string* \$actionSuffix)

Sets the default action suffix

abstract public **setDefaultNamespace** (*string* \$namespace)

Sets the default namespace

abstract public **setDefaultAction** (*string* \$actionName)

Sets the default action name

abstract public **setActionName** (*string* \$actionName)

Sets the action name to be dispatched

abstract public *string* **getActionName** ()

Gets last dispatched action name

abstract public **setParams** (*array* \$params)

Sets action params to be dispatched

abstract public *array* **getParams** ()

Gets action params

abstract public **setParam** (*mixed* \$param, *mixed* \$value)

Set a param by its name or numeric index

abstract public *mixed* **getParam** (*mixed* \$param, [*string/array* \$filters])

Gets a param by its name or numeric index

abstract public *boolean* **isFinished** ()

Checks if the dispatch loop is finished or has more pendent controllers/tasks to dispatch

abstract public *mixed* **getReturnValue** ()

Returns value returned by the lastest dispatched action

abstract public *object* **dispatch** ()

Dispatches a handle action taking into account the routing parameters

abstract public **forward** (*array* \$forward)

Fowards the execution flow to another controller/action

## 2.54.269 Interface Phalcon\EscaperInterface

Phalcon\EscaperInterface initializer

## Methods

abstract public **setEncoding** (*string* \$encoding)

Sets the encoding to be used by the escaper

abstract public *string* **getEncoding** ()

Returns the internal encoding used by the escaper

abstract public **setHtmlQuoteType** (*int* \$quoteType)

Sets the HTML quoting type for htmlspecialchars

abstract public *string* **escapeHtml** (*string* \$text)

Escapes a HTML string

abstract public *string* **escapeHtmlAttr** (*string* \$text)

Escapes a HTML attribute string

abstract public *string* **escapeCss** (*string* \$css)

Escape CSS strings by replacing non-alphanumeric chars by their hexadecimal representation

abstract public *string* **escapeJs** (*string* \$js)

Escape Javascript strings by replacing non-alphanumeric chars by their hexadecimal representation

abstract public *string* **escapeUrl** (*string* \$url)

Escapes a URL. Internally uses rawurlencode

## 2.54.270 Interface Phalcon\Events\EventsAwareInterface

Phalcon\Events\EventsAwareInterface initializer

### Methods

abstract public **setEventsManager** (*Phalcon\Events\ManagerInterface* \$eventsManager)

Sets the events manager

abstract public *Phalcon\Events\ManagerInterface* **getEventsManager** ()

Returns the internal event manager

## 2.54.271 Interface Phalcon\Events\ManagerInterface

Phalcon\Events\ManagerInterface initializer

### Methods

abstract public **attach** (*string* \$eventType, *object* \$handler)

Attach a listener to the events manager

abstract public **detachAll** ([*string* \$type])

Removes all events from the EventsManager

abstract public *mixed* **fire** (*string* \$eventType, *object* \$source, [*mixed* \$data])

Fires a event in the events manager causing that the active listeners will be notified about it

abstract public *array* **getListeners** (*string* \$type)

Returns all the attached listeners of a certain type

## 2.54.272 Interface Phalcon\FilterInterface

Phalcon\FilterInterface initializer

### Methods

abstract public *Phalcon\FilterInterface* **add** (*string* \$name, *callable* \$handler)

Adds a user-defined filter

abstract public *mixed* **sanitize** (*mixed* \$value, *mixed* \$filters)

Sanizites a value with a specified single or set of filters

abstract public *object[]* **getFilters** ()

Return the user-defined filters in the instance

## 2.54.273 Interface Phalcon\Filter\UserFilterInterface

Phalcon\Filter\UserFilterInterface initializer

### Methods

abstract public *mixed* **filter** (*mixed* \$value)

Filters a value

## 2.54.274 Interface Phalcon\FlashInterface

Phalcon\FlashInterface initializer

### Methods

abstract public *string* **error** (*string* \$message)

Shows a HTML error message

abstract public *string* **notice** (*string* \$message)

Shows a HTML notice/information message

abstract public *string* **success** (*string* \$message)

Shows a HTML success message

abstract public *string* **warning** (*string* \$message)

Shows a HTML warning message

abstract public *string* **message** (*string* \$type, *string* \$message)

Outputs a message

## 2.54.275 Interface Phalcon\Forms\ElementInterface

Phalcon\Forms\ElementInterface initializer

## Methods

abstract public *Phalcon\Forms\ElementInterface* **setForm** (*Phalcon\Forms\Form* \$form)

Sets the parent form to the element

abstract public *Phalcon\Forms\ElementInterface* **getForm** ()

Returns the parent form to the element

abstract public *Phalcon\Forms\ElementInterface* **setName** (*string* \$name)

Sets the element's name

abstract public *string* **getName** ()

Returns the element's name

abstract public *Phalcon\Forms\ElementInterface* **setFilters** (*array/string* \$filters)

Sets the element's filters

abstract public *Phalcon\Forms\ElementInterface* **addFilter** (*string* \$filter)

Adds a filter to current list of filters

abstract public *mixed* **getFilters** ()

Returns the element's filters

abstract public *Phalcon\Forms\ElementInterface* **addValidators** (*unknown* \$validators, [*unknown* \$merge])

Adds a group of validators

abstract public *Phalcon\Forms\ElementInterface* **addValidator** (*unknown* \$validator)

Adds a validator to the element

abstract public *Phalcon\Validation\ValidatorInterface* [] **getValidators** ()

Returns the validators registered for the element

abstract public *array* **prepareAttributes** ([*array* \$attributes], [*boolean* \$useChecked])

Returns an array of prepared attributes for Phalcon\Tag helpers according to the element's parameters

abstract public *Phalcon\Forms\ElementInterface* **setAttribute** (*string* \$attribute, *mixed* \$value)

Sets a default attribute for the element

abstract public *mixed* **getAttribute** (*string* \$attribute, [*mixed* \$defaultValue])

Returns the value of an attribute if present

abstract public *Phalcon\Forms\ElementInterface* **setAttributes** (*array* \$attributes)

Sets default attributes for the element

abstract public *array* **getAttributes** ()

Returns the default attributes for the element

abstract public *Phalcon\Forms\ElementInterface* **setUserOption** (*string* \$option, *mixed* \$value)

Sets an option for the element

abstract public *mixed* **getUserOption** (*string* \$option, [*mixed* \$defaultValue])

Returns the value of an option if present

abstract public *Phalcon\Forms\ElementInterface* **setUserOptions** (*array* \$options)

Sets options for the element

abstract public *array* **getUserOptions** ()

Returns the options for the element

abstract public *Phalcon\Forms\ElementInterface* **setLabel** (*string* \$label)

Sets the element label

abstract public *string* **getLabel** ()

Returns the element's label

abstract public *string* **label** ()

Generate the HTML to label the element

abstract public *Phalcon\Forms\ElementInterface* **setDefault** (*mixed* \$value)

Sets a default value in case the form does not use an entity or there is no value available for the element in *\$\_POST*

abstract public *mixed* **getDefault** ()

Returns the default value assigned to the element

abstract public *mixed* **getValue** ()

Returns the element's value

abstract public *Phalcon\Validation\Message\Group* **getMessages** ()

Returns the messages that belongs to the element The element needs to be attached to a form

abstract public *boolean* **hasMessages** ()

Checks whether there are messages attached to the element

abstract public *Phalcon\Forms\ElementInterface* **setMessages** (*Phalcon\Validation\Message\Group* \$group)

Sets the validation messages related to the element

abstract public *Phalcon\Forms\ElementInterface* **appendMessage** (*Phalcon\Validation\Message* \$message)

Appends a message to the internal message list

abstract public *Phalcon\Forms\Element* **clear** ()

Clears every element in the form to its default value

abstract public *string* **render** ([*array* \$attributes])

Renders the element widget

## 2.54.276 Interface Phalcon\Http\RequestInterface

Phalcon\Http\RequestInterface initializer

## Methods

abstract public *mixed* **get** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets a variable from the \$\_REQUEST superglobal applying filters if needed

abstract public *mixed* **getPost** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets a variable from the \$\_POST superglobal applying filters if needed

abstract public **getPut** ([*unknown* \$name], [*unknown* \$filters], [*unknown* \$defaultValue])

...

abstract public *mixed* **getQuery** ([*string* \$name], [*string/array* \$filters], [*mixed* \$defaultValue])

Gets variable from \$\_GET superglobal applying filters if needed

abstract public *mixed* **getServer** (*string* \$name)

Gets variable from \$\_SERVER superglobal

abstract public *boolean* **has** (*string* \$name)

Checks whether \$\_SERVER superglobal has certain index

abstract public *boolean* **hasPost** (*string* \$name)

Checks whether \$\_POST superglobal has certain index

abstract public **hasPut** (*unknown* \$name)

...

abstract public *boolean* **hasQuery** (*string* \$name)

Checks whether \$\_SERVER superglobal has certain index

abstract public *mixed* **hasServer** (*string* \$name)

Checks whether \$\_SERVER superglobal has certain index

abstract public *string* **getHeader** (*string* \$header)

Gets HTTP header from request data

abstract public *string* **getScheme** ()

Gets HTTP schema (http/https)

abstract public *boolean* **isAjax** ()

Checks whether request has been made using ajax. Checks if \$\_SERVER['HTTP\_X\_REQUESTED\_WITH']=='XMLHttpRequest'

abstract public *boolean* **isSoapRequested** ()

Checks whether request has been made using SOAP

abstract public *boolean* **isSecureRequest** ()

Checks whether request has been made using any secure layer

abstract public *string* **getRawBody** ()

Gets HTTP raws request body

abstract public *string* **getServerAddress** ()

Gets active server address IP

abstract public *string* **getServerName** ()

Gets active server name

**abstract public string getHttpHost ()**

Gets information about schema, host and port used by the request

**abstract public string getClientAddress ([boolean \$trustForwardedHeader])**

Gets most possibly client IPv4 Address. This methods search in `$_SERVER['REMOTE_ADDR']` and optionally in `$_SERVER['HTTP_X_FORWARDED_FOR']`

**abstract public string getMethod ()**

Gets HTTP method which request has been made

**abstract public string getUserAgent ()**

Gets HTTP user agent used to made the request

**abstract public boolean isMethod (string/array \$methods)**

Check if HTTP method match any of the passed methods

**abstract public boolean isPost ()**

Checks whether HTTP method is POST. if `$_SERVER['REQUEST_METHOD']=='POST'`

**abstract public boolean isGet ()**

Checks whether HTTP method is GET. if `$_SERVER['REQUEST_METHOD']=='GET'`

**abstract public boolean isPut ()**

Checks whether HTTP method is PUT. if `$_SERVER['REQUEST_METHOD']=='PUT'`

**abstract public boolean isHead ()**

Checks whether HTTP method is HEAD. if `$_SERVER['REQUEST_METHOD']=='HEAD'`

**abstract public boolean isDelete ()**

Checks whether HTTP method is DELETE. if `$_SERVER['REQUEST_METHOD']=='DELETE'`

**abstract public boolean isOptions ()**

Checks whether HTTP method is OPTIONS. if `$_SERVER['REQUEST_METHOD']=='OPTIONS'`

**abstract public boolean hasFiles ([boolean \$notErrored])**

Checks whether request include attached files

**abstract public Phalcon\Http\Request\FileInterface [] getUploadedFiles ([boolean \$notErrored])**

Gets attached files as Phalcon\HTTP\Request\FileInterface compatible instances

**abstract public string getHTTPReferer ()**

Gets web page that refers active request. ie: <http://www.google.com>

**abstract public array getAcceptableContent ()**

Gets array with mime/types and their quality accepted by the browser/client from `$_SERVER['HTTP_ACCEPT']`

**abstract public array getBestAccept ()**

Gets best mime/type accepted by the browser/client from `$_SERVER['HTTP_ACCEPT']`

**abstract public array getClientCharsets ()**

Gets charsets array and their quality accepted by the browser/client from `$_SERVER['HTTP_ACCEPT_CHARSET']`

abstract public `string getBestCharset ()`

Gets best charset accepted by the browser/client from `$_SERVER['HTTP_ACCEPT_CHARSET']`

abstract public `array getLanguages ()`

Gets languages array and their quality accepted by the browser/client from `$_SERVER['HTTP_ACCEPT_LANGUAGE']`

abstract public `string getBestLanguage ()`

Gets best language accepted by the browser/client from `$_SERVER['HTTP_ACCEPT_LANGUAGE']`

## 2.54.277 Interface Phalcon\Http\Request\FileInterface

Phalcon\Http\Request\FileInterface initializer

### Methods

abstract public `int getSize ()`

Returns the file size of the uploaded file

abstract public `string getName ()`

Returns the real name of the uploaded file

abstract public `string getTempName ()`

Returns the temporal name of the uploaded file

abstract public `string getType ()`

Returns the mime type reported by the browser This mime type is not completely secure, use `getRealType()` instead

abstract public `string getRealType ()`

Gets the real mime type of the upload file using finfo

abstract public `boolean moveTo (string $destination)`

Move the temporary file to a destination

## 2.54.278 Interface Phalcon\Http\ResponseInterface

Phalcon\Http\ResponseInterface initializer

### Methods

abstract public `Phalcon\Http\ResponseInterface setStatusCode (int $code, string $message)`

Sets the HTTP response code

abstract public `Phalcon\Http\Response\Headers getHeaders ()`

Returns headers set by the user

```
abstract public Phalcon\Http\ResponseInterface setHeader (string $name, string $value)
Overwrites a header in the response

abstract public Phalcon\Http\ResponseInterface setRawHeader (string $header)
Send a raw header to the response

abstract public Phalcon\Http\ResponseInterface resetHeaders ()
Resets all the stablished headers

abstract public Phalcon\Http\ResponseInterface setExpires (DateTime $datetime)
Sets output expire time header

abstract public Phalcon\Http\ResponseInterface setNotModified ()
Sends a Not-Modified response

abstract public Phalcon\Http\ResponseInterface setContent-Type (string $contentType, [string $charset])
Sets the response content-type mime, optionally the charset

abstract public Phalcon\Http\ResponseInterface redirect ([string $location], [boolean $externalRedirect], [int $statusCode])
Redirect by HTTP to another action or URL

abstract public Phalcon\Http\ResponseInterface setContent (string $content)
Sets HTTP response body

abstract public Phalcon\Http\ResponseInterface setJsonContent (string $content)
Sets HTTP response body. The parameter is automatically converted to JSON

<?php
$response->setJsonContent(array("status" => "OK"));

abstract public Phalcon\Http\ResponseInterface appendContent (string $content)
Appends a string to the HTTP response body

abstract public string getContent ()
Gets the HTTP response body

abstract public Phalcon\Http\ResponseInterface sendHeaders ()
Sends headers to the client

abstract public Phalcon\Http\ResponseInterface sendCookies ()
Sends cookies to the client

abstract public Phalcon\Http\ResponseInterface send ()
Prints out HTTP response to the client

abstract public setFileToSend (string $filePath, [string $attachmentName])
Sets an attached file to be sent at the end of the request
```

## 2.54.279 Interface Phalcon\Http\Response\CookiesInterface

Phalcon\Http\Response\CookiesInterface initializer

## Methods

abstract public *Phalcon\Http\Response\CookiesInterface* **useEncryption** (*boolean \$useEncryption*)

Set if cookies in the bag must be automatically encrypted/decrypted

abstract public *boolean isUsingEncryption* ()

Returns if the bag is automatically encrypting/decrypting cookies

abstract public *Phalcon\Http\Response\CookiesInterface* **set** (*string \$name*, [*mixed \$value*], [*int \$expire*], [*string \$path*], [*boolean \$secure*], [*string \$domain*], [*boolean \$httpOnly*])

Sets a cookie to be sent at the end of the request

abstract public *Phalcon\Http\Cookie* **get** (*string \$name*)

Gets a cookie from the bag

abstract public *boolean has* (*string \$name*)

Check if a cookie is defined in the bag or exists in the `$_COOKIE` superglobal

abstract public *boolean delete* (*string \$name*)

Deletes a cookie by its name This method does not removes cookies from the `$_COOKIE` superglobal

abstract public *boolean send* ()

Sends the cookies to the client

abstract public *Phalcon\Http\Response\CookiesInterface* **reset** ()

Reset set cookies

## 2.54.280 Interface Phalcon\Http\Response\HeadersInterface

Phalcon\Http\Response\HeadersInterface initializer

## Methods

abstract public **set** (*string \$name*, *string \$value*)

Sets a header to be sent at the end of the request

abstract public *string get* (*string \$name*)

Gets a header value from the internal bag

abstract public **setRaw** (*string \$header*)

Sets a raw header to be sent at the end of the request

abstract public *boolean send* ()

Sends the headers to the client

abstract public **reset** ()

Reset set headers

abstract public *array toArray* ()

Returns the current headers as an array

## 2.54.281 Interface Phalcon\Image\AdapterInterface

Phalcon\Image\AdapterInterface initializer

### Methods

abstract public *Phalcon\Image\Adapter* **resize** ([*unknown \$width*, [*unknown \$height*], [*unknown \$master*]])

Resize the image to the given size. Either the width or the height can be omitted and the image will be resized proportionally.

abstract public **liquidRescale** (*unknown \$width*, *unknown \$height*, [*unknown \$delta\_x*], [*unknown \$rigidity*])

...

abstract public *Phalcon\Image\Adapter* **crop** (*unknown \$width*, *unknown \$height*, [*unknown \$offset\_x*], [*unknown \$offset\_y*])

Crop an image to the given size. Either the width or the height can be omitted and the current width or height will be used.

abstract public *Phalcon\Image\Adapter* **rotate** (*unknown \$degrees*)

Rotate the image by a given amount.

abstract public *Phalcon\Image\Adapter* **flip** (*unknown \$direction*)

Flip the image along the horizontal or vertical axis.

abstract public *Phalcon\Image\Adapter* **sharpen** (*unknown \$amount*)

Sharpen the image by a given amount.

abstract public *Phalcon\Image\Adapter* **reflection** ([*unknown \$height*], [*unknown \$opacity*], [*unknown \$fade\_in*])

Add a reflection to an image. The most opaque part of the reflection will be equal to the opacity setting and fade out to full transparent. Alpha transparency is preserved.

abstract public *Phalcon\Image\Adapter* **watermark** (*unknown \$watermark*, [*unknown \$offset\_x*], [*unknown \$offset\_y*], [*unknown \$opacity*])

Add a watermark to an image with a specified opacity. Alpha transparency will be preserved.

abstract public **text** (*unknown \$text*, [*unknown \$offset\_x*], [*unknown \$offset\_y*], [*unknown \$opacity*], [*unknown \$color*], [*unknown \$size*], [*unknown \$fontfile*])

...

abstract public **mask** (*unknown \$mask*)

...

abstract public *Phalcon\Image\Adapter* **background** (*unknown \$color*, [*unknown \$quality*])

Set the background color of an image. This is only useful for images with alpha transparency.

abstract public **blur** ([*unknown \$radius*])

...

abstract public **pixelate** ([*unknown \$amount*])

...

abstract public *Phalcon\Image\Adapter* **save** ([*unknown* \$file], [*unknown* \$quality])

Save the image. If the filename is omitted, the original image will be overwritten.

abstract public *Phalcon\Image\Adapter* **render** ([*unknown* \$type], [*unknown* \$quality])

Render the image and return the binary string.

## 2.54.282 Interface Phalcon\Logger\AdapterInterface

Phalcon\Logger\AdapterInterface initializer

### Methods

abstract public *Phalcon\Logger\Adapter* **setFormatter** (*Phalcon\Logger\FormatterInterface* \$formatter)

Sets the message formatter

abstract public *Phalcon\Logger\FormatterInterface* **getFormatter** ()

Returns the internal formatter

abstract public *Phalcon\Logger\Adapter* **setLogLevel** (*int* \$level)

Filters the logs sent to the handlers to be greater or equals than a specific level

abstract public *int* **getLogLevel** ()

Returns the current log level

abstract public *Phalcon\Logger\Adapter* **begin** ()

Starts a transaction

abstract public *Phalcon\Logger\Adapter* **commit** ()

Commits the internal transaction

abstract public *Phalcon\Logger\Adapter* **rollback** ()

Rollbacks the internal transaction

abstract public *boolean* **close** ()

Closes the logger

abstract public *Phalcon\Logger\AdapterInterface* **log** (*int/string* \$type, *string* \$message, [*array* \$context])

Sends/Writes messages to the file log

abstract public *Phalcon\Logger\AdapterInterface* **debug** (*string* \$message, [*array* \$context])

Sends/Writes a debug message to the log

abstract public *Phalcon\Logger\AdapterInterface* **info** (*string* \$message, [*array* \$context])

Sends/Writes an info message to the log

abstract public *Phalcon\Logger\AdapterInterface* **notice** (*string* \$message, [*unknown* \$context])

Sends/Writes a notice message to the log

abstract public *Phalcon\Logger\AdapterInterface* **warning** (*string* \$message, [*array* \$context])

Sends/Writes a warning message to the log

abstract public *Phalcon\Logger\AdapterInterface* **error** (*string* \$message, [*array* \$context])

Sends/Writes an error message to the log

abstract public *Phalcon\Logger\AdapterInterface* **critical** (*string \$message, [array \$context]*)

Sends/Writes a critical message to the log

abstract public *Phalcon\Logger\AdapterInterface* **alert** (*string \$message, [array \$context]*)

Sends/Writes an alert message to the log

abstract public *Phalcon\Logger\AdapterInterface* **emergency** (*string \$message, [array \$context]*)

Sends/Writes an emergency message to the log

## 2.54.283 Interface Phalcon\Logger\FormatterInterface

Phalcon\Logger\FormatterInterface initializer

### Methods

abstract public **format** (*string \$message, int \$type, int \$timestamp, array \$context*)

Applies a format to a message before sent it to the internal log

## 2.54.284 Interface Phalcon\Mvc\CollectionInterface

Phalcon\Mvc\CollectionInterface initializer

### Methods

abstract public **setId** (*mixed \$id*)

Sets a value for the `_id` property, creates a `MongoId` object if needed

abstract public *MongoId* **getId** ()

Returns the value of the `_id` property

abstract public *array* **getReservedAttributes** ()

Returns an array with reserved properties that cannot be part of the insert/update

abstract public *string* **getSource** ()

Returns collection name mapped in the model

abstract public **setConnectionService** (*string \$connectionService*)

Sets a service in the services container that returns the Mongo database

abstract public *MongoDb* **getConnection** ()

Retrieves a database connection

abstract public *mixed* **readAttribute** (*string \$attribute*)

Reads an attribute value by its name

abstract public **writeAttribute** (*string \$attribute, mixed \$value*)

Writes an attribute value by its name

abstract public static *Phalcon\ Mvc\ Collection* **cloneResult** (*Phalcon\ Mvc\ Collection* \$collection, *array* \$document)

Returns a cloned collection

abstract public *boolean* **fireEvent** (*string* \$eventName)

Fires an event, implicitly calls behaviors and listeners in the events manager are notified

abstract public *boolean* **fireEventCancel** (*string* \$eventName)

Fires an event, implicitly listeners in the events manager are notified This method stops if one of the callbacks/listeners returns boolean false

abstract public *boolean* **validationHasFailed** ()

Check whether validation process has generated any messages

abstract public *Phalcon\ Mvc\ Model\ MessageInterface* [] **getMessages** ()

Returns all the validation messages

abstract public **appendMessage** (*Phalcon\ Mvc\ Model\ MessageInterface* \$message)

Appends a customized message on the validation process

abstract public *boolean* **save** ()

Creates/Updates a collection based on the values in the attributes

abstract public static *Phalcon\ Mvc\ Collection* **findById** (*string* \$id)

Find a document by its id

abstract public static *array* **findFirst** (*[array* \$parameters])

Allows to query the first record that match the specified conditions

abstract public static *array* **find** (*[array* \$parameters])

Allows to query a set of records that match the specified conditions

abstract public static *array* **count** (*[array* \$parameters])

Perform a count over a collection

abstract public *boolean* **delete** ()

Deletes a model instance. Returning true on success or false otherwise

## 2.54.285 Interface Phalcon\ Mvc\ Collection\ ManagerInterface

Phalcon\ Mvc\ Collection\ ManagerInterface initializer

### Methods

abstract public **setCustomEventsManager** (*Phalcon\ Mvc\ CollectionInterface* \$model, *Phalcon\ Events\ ManagerInterface* \$eventsManager)

Sets a custom events manager for a specific model

abstract public *Phalcon\ Events\ ManagerInterface* **getCustomEventsManager** (*Phalcon\ Mvc\ CollectionInterface* \$model)

Returns a custom events manager related to a model

```
abstract public initialize (Phalcon\ Mvc\ CollectionInterface $model)
Initializes a model in the models manager

abstract public bool isInitialized (string $modelName)
Check whether a model is already initialized

abstract public Phalcon\ Mvc\ CollectionInterface getLastInitialized ()
Get the latest initialized model

abstract public setConnectionService (Phalcon\ Mvc\ CollectionInterface $model, string $connectionService)
Sets a connection service for a specific model

abstract public useImplicitObjectIds (Phalcon\ Mvc\ CollectionInterface $model, boolean $useImplicitObjectIds)
Sets if a model must use implicit objects ids

abstract public boolean isUsingImplicitObjectIds (Phalcon\ Mvc\ CollectionInterface $model)
Checks if a model is using implicit object ids

abstract public Phalcon\ Db\ AdapterInterface getConnection (Phalcon\ Mvc\ CollectionInterface $model)
Returns the connection related to a model

abstract public notifyEvent (string $eventName, Phalcon\ Mvc\ CollectionInterface $model)
Receives events generated in the models and dispatches them to a events-manager if available Notify the behaviors that are listening in the model
```

## 2.54.286 Interface Phalcon\ Mvc\ ControllerInterface

Phalcon\ Mvc\ ControllerInterface initializer

## 2.54.287 Interface Phalcon\ Mvc\ DispatcherInterface

*extends* PhalconDispatcherInterface

Phalcon\ Mvc\ DispatcherInterface initializer

### Methods

```
abstract public setControllerSuffix (string $controllerSuffix)
Sets the default controller suffix

abstract public setDefaultController (string $controllerName)
Sets the default controller name

abstract public setControllerName (string $controllerName, [unknown $isExact])
Sets the controller name to be dispatched

abstract public string getControllerName ()
Gets last dispatched controller name
```

abstract public *Phalcon\ Mvc\ ControllerInterface* **getLastController** ()

Returns the lastest dispatched controller

abstract public *Phalcon\ Mvc\ ControllerInterface* **getActiveController** ()

Returns the active controller in the dispatcher

abstract public **setActionSuffix** (*string \$actionSuffix*) inherited from Phalcon\DispatcherInterface

Sets the default action suffix

abstract public **setDefaultNamespace** (*string \$namespace*) inherited from Phalcon\DispatcherInterface

Sets the default namespace

abstract public **setDefaultAction** (*string \$actionName*) inherited from Phalcon\DispatcherInterface

Sets the default action name

abstract public **setActionName** (*string \$actionName*) inherited from Phalcon\DispatcherInterface

Sets the action name to be dispatched

abstract public *string* **getActionName** () inherited from Phalcon\DispatcherInterface

Gets last dispatched action name

abstract public **setParams** (*array \$params*) inherited from Phalcon\DispatcherInterface

Sets action params to be dispatched

abstract public *array* **getParams** () inherited from Phalcon\DispatcherInterface

Gets action params

abstract public **setParam** (*mixed \$param, mixed \$value*) inherited from Phalcon\DispatcherInterface

Set a param by its name or numeric index

abstract public *mixed* **getParam** (*mixed \$param, [string/array \$filters]*) inherited from Phalcon\DispatcherInterface

Gets a param by its name or numeric index

abstract public *boolean* **isFinished** () inherited from Phalcon\DispatcherInterface

Checks if the dispatch loop is finished or has more pendent controllers/tasks to disptach

abstract public *mixed* **getReturnValue** () inherited from Phalcon\DispatcherInterface

Returns value returned by the lastest dispatched action

abstract public *object* **dispatch** () inherited from Phalcon\DispatcherInterface

Dispatches a handle action taking into account the routing parameters

abstract public **forward** (*array \$forward*) inherited from Phalcon\DispatcherInterface

Forwards the execution flow to another controller/action

## 2.54.288 Interface Phalcon\ Mvc\ Micro\ CollectionInterface

Phalcon\ Mvc\ Micro\ CollectionInterface initializer

## Methods

abstract public *Phalcon\|Mvc\|Micro\|Collection* **setPrefix** (*string* \$prefix)

Sets a prefix for all routes added to the collection

abstract public *string* **getPrefix** ()

Returns the collection prefix if any

abstract public *array* **getHandlers** ()

Returns the registered handlers

abstract public *Phalcon\|Mvc\|Micro\|Collection* **setHandler** (*mixed* \$handler, [*boolean* \$lazy])

Sets the main handler

abstract public *Phalcon\|Mvc\|Micro\|Collection* **setLazy** (*boolean* \$lazy)

Sets if the main handler must be lazy loaded

abstract public *boolean* **isLazy** ()

Returns if the main handler must be lazy loaded

abstract public *mixed* **getHandler** ()

Returns the main handler

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **map** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **get** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is GET

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **post** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is POST

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **put** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is PUT

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **patch** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is PATCH

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **head** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is HEAD

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **delete** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is DELETE

abstract public *Phalcon\|Mvc\|Router\|RouteInterface* **options** (*string* \$routePattern, *callable* \$handler)

Maps a route to a handler that only matches if the HTTP method is OPTIONS

### 2.54.289 Interface Phalcon\Mvc\Micro\MiddlewareInterface

Phalcon\Mvc\Micro\MiddlewareInterface initializer

## Methods

abstract public **call** (*Phalcon\ Mvc\ Micro \$application*)

Calls the middleware

### 2.54.290 Interface Phalcon\ Mvc\ ModelInterface

Phalcon\ Mvc\ ModelInterface initializer

## Methods

abstract public *Phalcon\ Mvc\ ModelInterface* **setTransaction** (*Phalcon\ Mvc\ Model\ TransactionInterface \$transaction*)

Sets a transaction related to the Model instance

abstract public *string* **getSource** ()

Returns table name mapped in the model

abstract public *string* **getSchema** ()

Returns schema name where table mapped is located

abstract public **setConnectionService** (*string \$connectionService*)

Sets both read/write connection services

abstract public **setWriteConnectionService** (*string \$connectionService*)

Sets the DependencyInjection connection service used to write data

abstract public **setReadConnectionService** (*string \$connectionService*)

Sets the DependencyInjection connection service used to read data

abstract public *string* **getReadConnectionService** ()

Returns DependencyInjection connection service used to read data

abstract public *string* **getWriteConnectionService** ()

Returns DependencyInjection connection service used to write data

abstract public *Phalcon\ Db\ AdapterInterface* **getReadConnection** ()

Gets internal database connection

abstract public *Phalcon\ Db\ AdapterInterface* **getWriteConnection** ()

Gets internal database connection

abstract public *Phalcon\ Mvc\ Model* **assign** (*array \$data, [array \$columnMap]*)

Assigns values to a model from an array

abstract public static *Phalcon\ Mvc\ Model* **\$result cloneResultMap** (*Phalcon\ Mvc\ Model \$base, array \$data, array \$columnMap, [int \$dirtyState], [boolean \$keepSnapshots]*)

Assigns values to a model from an array returning a new model

abstract public static *Phalcon\ Mvc\ Model* **cloneResult** (*Phalcon\ Mvc\ Model \$base, array \$data, [int \$dirtyState]*)

Assigns values to a model from an array returning a new model

`abstract public static cloneResultMapHydrate (array $data, array $columnMap, int $hydrationMode)`

Returns an hydrated result based on the data and the column map

`abstract public static Phalcon\|Mvc\|Model\|ResultsetInterface find ([array $parameters])`

Allows to query a set of records that match the specified conditions

`abstract public static Phalcon\|Mvc\|ModelInterface findFirst ([array $parameters])`

Allows to query the first record that match the specified conditions

`abstract public static Phalcon\|Mvc\|Model\|CriteriaInterface query ([Phalcon\|DiInterface $dependencyInjector])`

Create a criteria for a especific model

`abstract public static int count ([array $parameters])`

Allows to count how many records match the specified conditions

`abstract public static double sum ([array $parameters])`

Allows to calculate a summatory on a column that match the specified conditions

`abstract public static mixed maximum ([array $parameters])`

Allows to get the maximum value of a column that match the specified conditions

`abstract public static mixed minimum ([array $parameters])`

Allows to get the minimum value of a column that match the specified conditions

`abstract public static double average ([array $parameters])`

Allows to calculate the average value on a column matching the specified conditions

`abstract public boolean fireEvent (string $eventName)`

Fires an event, implicitly calls behaviors and listeners in the events manager are notified

`abstract public boolean fireEventCancel (string $eventName)`

Fires an event, implicitly calls behaviors and listeners in the events manager are notified This method stops if one of the callbacks/listeners returns boolean false

`abstract public appendMessage (Phalcon\|Mvc\|Model\|MessageInterface $message)`

Appends a customized message on the validation process

`abstract public boolean validationHasFailed ()`

Check whether validation process has generated any messages

`abstract public Phalcon\|Mvc\|Model\|MessageInterface [] getMessages ([unknown $filter])`

Returns all the validation messages

`abstract public boolean save ([array $data], [array $whiteList])`

Inserts or updates a model instance. Returning true on success or false otherwise.

`abstract public boolean create ([array $data], [array $whiteList])`

Inserts a model instance. If the instance already exists in the persistance it will throw an exception Returning true on success or false otherwise.

`abstract public boolean update ([array $data], [array $whiteList])`

Updates a model instance. If the instance doesn't exist in the persistance it will throw an exception Returning true on success or false otherwise.

abstract public **boolean delete ()**

Deletes a model instance. Returning true on success or false otherwise.

abstract public **int getOperationMade ()**

Returns the type of the latest operation performed by the ORM Returns one of the OP\_\* class constants

abstract public **refresh ()**

Refreshes the model attributes re-querying the record from the database

abstract public **mixed readAttribute (string \$attribute)**

Reads an attribute value by its name

abstract public **writeAttribute (string \$attribute, mixed \$value)**

Writes an attribute value by its name

abstract public **Phalcon\ Mvc\ Model\ ResultsetInterface getRelated (string \$alias, [array \$arguments])**

Returns related records based on defined relations

## 2.54.291 Interface Phalcon\ Mvc\ Model\ BehaviorInterface

Phalcon\ Mvc\ Model\ BehaviorInterface initializer

### Methods

abstract public **notify (string \$type, Phalcon\ Mvc\ ModelInterface \$model)**

This method receives the notifications from the EventsManager

abstract public **missingMethod (Phalcon\ Mvc\ ModelInterface \$model, string \$method, [array \$arguments])**

Calls a method when it's missing in the model

## 2.54.292 Interface Phalcon\ Mvc\ Model\ CriterialInterface

Phalcon\ Mvc\ Model\ CriteriaInterface initializer

### Methods

abstract public **Phalcon\ Mvc\ Model\ CriteriaInterface setModelName (string \$modelName)**

Set a model on which the query will be executed

abstract public **string getModelName ()**

Returns an internal model name on which the criteria will be applied

abstract public **Phalcon\ Mvc\ Model\ CriteriaInterface bind (string \$bindParams)**

Adds the bind parameter to the criteria

abstract public **Phalcon\ Mvc\ Model\ Criteria bindTypes (string \$bindTypes)**

Sets the bind types in the criteria This method replaces all previously set bound parameters

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **columns** (*string/array* \$columns)

Sets the columns to be queried

<?php

```
$criteria->columns(array('id', 'name'));
```

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **join** (*string* \$model, [*string* \$conditions], [*string* \$alias], [*string* \$type])

Adds a join to the query

<?php

```
$criteria->join('Robots');
$criteria->join('Robots', 'r.id = RobotsParts.robots_id');
$criteria->join('Robots', 'r.id = RobotsParts.robots_id', 'r');
$criteria->join('Robots', 'r.id = RobotsParts.robots_id', 'r', 'LEFT');
```

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **where** (*string* \$conditions)

Adds the conditions parameter to the criteria

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **conditions** (*string* \$conditions)

Adds the conditions parameter to the criteria

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **orderBy** (*string* \$orderColumns)

Adds the order-by parameter to the criteria

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **limit** (*int* \$limit, [*int* \$offset])

Sets the limit parameter to the criteria

abstract public *Phalcon|Mvc|Model|CriteriaInterface* **forUpdate** ([*boolean* \$forUpdate])

Sets the “for\_update” parameter to the criteria

abstract public *Phalcon|Mvc|Model|Criteria* **sharedLock** ([*boolean* \$sharedLock])

Sets the “shared\_lock” parameter to the criteria

abstract public *Phalcon|Mvc|Model|Criteria* **andWhere** (*string* \$conditions, [*array* \$bindParams], [*array* \$bindTypes])

Appends a condition to the current conditions using an AND operator

abstract public *Phalcon|Mvc|Model|Criteria* **orWhere** (*string* \$conditions, [*array* \$bindParams], [*array* \$bindTypes])

Appends a condition to the current conditions using an OR operator

abstract public *Phalcon|Mvc|Model|Query|Builder* **betweenWhere** (*string* \$expr, *mixed* \$minimum, *mixed* \$maximum)

Appends a BETWEEN condition to the current conditions

<?php

```
$criteria->betweenWhere('price', 100.25, 200.50);
```

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **notBetweenWhere** (*string \$expr, mixed \$minimum, mixed \$maximum*)

Appends a NOT BETWEEN condition to the current conditions

<?php

```
$criteria->notBetweenWhere('price', 100.25, 200.50);
```

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **inWhere** (*string \$expr, array \$values*)

Appends an IN condition to the current conditions

<?php

```
$criteria->inWhere('id', [1, 2, 3]);
```

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **notInWhere** (*string \$expr, array \$values*)

Appends a NOT IN condition to the current conditions

<?php

```
$criteria->notInWhere('id', [1, 2, 3]);
```

abstract public *string getWhere* ()

Returns the conditions parameter in the criteria

abstract public *string getConditions* ()

Returns the conditions parameter in the criteria

abstract public *string getLimit* ()

Returns the limit parameter in the criteria

abstract public *string getOrder* ()

Returns the order parameter in the criteria

abstract public *string getParams* ()

Returns all the parameters defined in the criteria

abstract public static *static fromInput* (*Phalcon\ DiInterface \$dependencyInjector, string \$modelName, array \$data*)

Builds a Phalcon\ Mvc\ Model\ Criteria based on an input array like \$\_POST

abstract public *Phalcon\ Mvc\ Model\ ResultsetInterface execute* ()

Executes a find using the parameters built with the criteria

## 2.54.293 Interface Phalcon\ Mvc\ Model\ ManagerInterface

Phalcon\ Mvc\ Model\ ManagerInterface initializer

### Methods

abstract public **initialize** (*Phalcon\ Mvc\ ModelInterface \$model*)

Initializes a model in the model manager

abstract public *boolean* **isInitialized** (*string* \$modelName)

Check of a model is already initialized

abstract public *Phalcon\ Mvc\ ModelInterface* **getLastInitialized** ()

Get last initialized model

abstract public *Phalcon\ Mvc\ ModelInterface* **load** (*string* \$modelName, *boolean* \$newInstance)

Loads a model throwing an exception if it doesn't exist

abstract public *Phalcon\ Mvc\ Model\ RelationInterface* **addHasOne** (*Phalcon\ Mvc\ ModelInterface* \$model, *mixed* \$fields, *string* \$referencedModel, *mixed* \$referencedFields, [*array* \$options])

Setup a 1-1 relation between two models

abstract public *Phalcon\ Mvc\ Model\ RelationInterface* **addBelongsTo** (*Phalcon\ Mvc\ ModelInterface* \$model, *mixed* \$fields, *string* \$referencedModel, *mixed* \$referencedFields, [*array* \$options])

Setup a relation reverse 1-1 between two models

abstract public *Phalcon\ Mvc\ Model\ RelationInterface* **addHasMany** (*Phalcon\ Mvc\ ModelInterface* \$model, *mixed* \$fields, *string* \$referencedModel, *mixed* \$referencedFields, [*array* \$options])

Setup a relation 1-n between two models

abstract public *boolean* **existsBelongsTo** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has a belongsTo relation with another model

abstract public *boolean* **existsHasMany** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has ahasMany relation with another model

abstract public *boolean* **existsHasOne** (*string* \$modelName, *string* \$modelRelation)

Checks whether a model has ahasOne relation with another model

abstract public *Phalcon\ Mvc\ Model\ ResultsetInterface* **getBelongsToRecords** (*string* \$method, *string* \$modelName, *string* \$modelRelation, *Phalcon\ Mvc\ Model* \$record, [*array* \$parameters])

Gets belongsTo related records from a model

abstract public *Phalcon\ Mvc\ Model\ ResultsetInterface* **getHasManyRecords** (*string* \$method, *string* \$modelName, *string* \$modelRelation, *Phalcon\ Mvc\ Model* \$record, [*array* \$parameters])

GetshasMany related records from a model

abstract public *Phalcon\ Mvc\ Model\ ResultsetInterface* **getHasOneRecords** (*string* \$method, *string* \$modelName, *string* \$modelRelation, *Phalcon\ Mvc\ Model* \$record, [*array* \$parameters])

Gets belongsTo related records from a model

abstract public *array* **getBelongsTo** (*Phalcon\ Mvc\ ModelInterface* \$model)

Gets belongsTo relations defined on a model

abstract public *array* **getHasMany** (*Phalcon\ Mvc\ ModelInterface* \$model)

GetshasMany relations defined on a model

abstract public *array* **getHasOne** (*Phalcon\ Mvc\ ModelInterface* \$model)

GetshasOne relations defined on a model

abstract public *array* **getHasOneAndHasMany** (*Phalcon\ Mvc\ ModelInterface* \$model)

GetshasOne relations defined on a model

abstract public *Phalcon\ Mvc\ Model\ RelationInterface* [] **getRelations** (*string \$modelName*)

Query all the relationships defined on a model

abstract public *array getRelationsBetween* (*string \$first, string \$second*)

Query the relations between two models

abstract public *Phalcon\ Mvc\ Model\ QueryInterface* **createQuery** (*string \$phql*)

Creates a Phalcon\ Mvc\ Model\ Query without execute it

abstract public *Phalcon\ Mvc\ Model\ QueryInterface* **executeQuery** (*string \$phql, [array \$placeholders]*)

Creates a Phalcon\ Mvc\ Model\ Query and execute it

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **createBuilder** ([*string \$params*])

Creates a Phalcon\ Mvc\ Model\ Query\ Builder

abstract public **addBehavior** (*Phalcon\ Mvc\ ModelInterface \$model, Phalcon\ Mvc\ Model\ BehaviorInterface \$behavior*)

Binds a behavior to a model

abstract public **notifyEvent** (*string \$eventName, Phalcon\ Mvc\ ModelInterface \$model*)

Receives events generated in the models and dispatches them to a events-manager if available Notify the behaviors that are listening in the model

abstract public *boolean missingMethod* (*Phalcon\ Mvc\ ModelInterface \$model, string \$eventName, array \$data*)

Dispatch a event to the listeners and behaviors This method expects that the endpoint listeners/behaviors returns true meaning that a least one is implemented

abstract public *Phalcon\ Mvc\ Model\ QueryInterface* **getLastQuery** ()

Returns the last query created or executed in the

## 2.54.294 Interface Phalcon\ Mvc\ Model\ MessageInterface

Phalcon\ Mvc\ Model\ MessageInterface initializer

### Methods

abstract public **setType** (*string \$type*)

Sets message type

abstract public *string getType* ()

Returns message type

abstract public **setMessage** (*string \$message*)

Sets verbose message

abstract public *string getMessage* ()

Returns verbose message

abstract public **setField** (*string \$field*)

Sets field name related to message

abstract public *string* **getField** ()

Returns field name related to message

## 2.54.295 Interface Phalcon\Mvc\Model\MetaDataInterface

Phalcon\Mvc\Model\MetaDataInterface initializer

### Methods

abstract public **setStrategy** (*Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection* \$strategy)

Set the meta-data extraction strategy

abstract public *Phalcon\ Mvc\ Model\ MetaData\ Strategy\ Introspection* **getStrategy** ()

Return the strategy to obtain the meta-data

abstract public *array* **readMetaData** (*Phalcon\ Mvc\ ModelInterface* \$model)

Reads meta-data for certain model

abstract public *mixed* **readMetaDataIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, *int* \$index)

Reads meta-data for certain model using a MODEL\_\* constant

abstract public **writeMetaDataIndex** (*Phalcon\ Mvc\ Model* \$model, *int* \$index, *mixed* \$data, *unknown* \$replace)

Writes meta-data for certain model using a MODEL\_\* constant

abstract public *array* **readColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model)

Reads the ordered/reversed column map for certain model

abstract public **readColumnMapIndex** (*Phalcon\ Mvc\ ModelInterface* \$model, *int* \$index)

Reads column-map information for certain model using a MODEL\_\* constant

abstract public *array* **getAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns table attributes names (fields)

abstract public *array* **getPrimaryKeyAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns an array of fields which are part of the primary key

abstract public *array* **getNonPrimaryKeyAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns an arrau of fields which are not part of the primary key

abstract public *array* **getNotNullAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns an array of not null attributes

abstract public *array* **getDataTypes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns attributes and their data types

abstract public *array* **getDataTypesNumeric** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns attributes which types are numerical

abstract public *string* **getIdentityField** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the name of identity field (if one is present)

abstract public *array* **getBindTypes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns attributes and their bind data types

abstract public *array* **getAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns attributes that must be ignored from the INSERT SQL generation

abstract public *array* **getAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns attributes that must be ignored from the UPDATE SQL generation

abstract public **setAutomaticCreateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace)

Set the attributes that must be ignored from the INSERT SQL generation

abstract public **setAutomaticUpdateAttributes** (*Phalcon\ Mvc\ ModelInterface* \$model, *array* \$attributes, *unknown* \$replace)

Set the attributes that must be ignored from the UPDATE SQL generation

abstract public *array* **getColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the column map if any

abstract public *array* **getReverseColumnMap** (*Phalcon\ Mvc\ ModelInterface* \$model)

Returns the reverse column map if any

abstract public *boolean* **hasAttribute** (*Phalcon\ Mvc\ ModelInterface* \$model, *string* \$attribute)

Check if a model has certain attribute

abstract public *boolean* **isEmpty** ()

Checks if the internal meta-data container is empty

abstract public **reset** ()

Resets internal meta-data in order to regenerate it

abstract public *array* **read** (*string* \$key)

Reads meta-data from the adapter

abstract public **write** (*string* \$key, *array* \$data)

Writes meta-data to the adapter

## 2.54.296 Interface Phalcon\ Mvc\ Model\ QueryInterface

Phalcon\ Mvc\ Model\ QueryInterface initializer

### Methods

abstract public *array* **parse** ()

Parses the intermediate code produced by Phalcon\ Mvc\ Model\ Query\ Lang generating another intermediate representation that could be executed by Phalcon\ Mvc\ Model\ Query

abstract public *mixed* **execute** ([*array* \$bindParams], [*array* \$bindTypes])

Executes a parsed PHQL statement

## 2.54.297 Interface Phalcon\Mvc\Model\Query\BuilderInterface

Phalcon\Mvc\Model\Query\BuilderInterface initializer

### Methods

abstract public **distinct** (*unknown* \$distinct)

...

abstract public **getDistinct** ()

...

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **columns** (*string/array* \$columns)

Sets the columns to be queried

abstract public *string/array* **getColumns** ()

Return the columns to be queried

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **from** (*string/array* \$models)

Sets the models who makes part of the query

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **addFrom** (*string* \$model, [*string* \$alias])

Add a model to take part of the query

abstract public *string/array* **getFrom** ()

Return the models who makes part of the query

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **join** (*string* \$model, [*string* \$conditions], [*string* \$alias])

Adds a INNER join to the query

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **innerJoin** (*string* \$model, [*string* \$conditions], [*string* \$alias])

Adds a INNER join to the query

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **leftJoin** (*string* \$model, [*string* \$conditions], [*string* \$alias])

Adds a LEFT join to the query

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **rightJoin** (*string* \$model, [*string* \$conditions], [*string* \$alias])

Adds a RIGHT join to the query

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **where** (*string* \$conditions, [*array* \$bindParams], [*array* \$bindTypes])

Sets conditions for the query

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **andWhere** (*string* \$conditions, [*array* \$bindParams], [*array* \$bindTypes])

Appends a condition to the current conditions using a AND operator

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **orWhere** (*string* \$conditions, [*array* \$bindParams], [*array* \$bindTypes])

Appends a condition to the current conditions using a OR operator

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **betweenWhere** (*string \$expr, mixed \$minimum, mixed \$maximum*)

Appends a BETWEEN condition to the current conditions

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **notBetweenWhere** (*string \$expr, mixed \$minimum, mixed \$maximum*)

Appends a NOT BETWEEN condition to the current conditions

<?php

```
$builder->notBetweenWhere('price', 100.25, 200.50);
```

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **inWhere** (*string \$expr, array \$values*)

Appends an IN condition to the current conditions

abstract public *Phalcon\ Mvc\ Model\ Query\ Builder* **notInWhere** (*string \$expr, array \$values*)

Appends a NOT IN condition to the current conditions

abstract public *string/array* **getWhere** ()

Return the conditions for the query

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **orderBy** (*string \$orderBy*)

Sets a ORDER BY condition clause

abstract public *string/array* **getOrderBy** ()

Return the set ORDER BY clause

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **having** (*string \$having*)

Sets a HAVING condition clause

abstract public *string/array* **getHaving** ()

Returns the HAVING condition clause

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **limit** (*int \$limit, [int \$offset]*)

Sets a LIMIT clause

abstract public *string/array* **getLimit** ()

Returns the current LIMIT clause

abstract public *Phalcon\ Mvc\ Model\ Query\ BuilderInterface* **groupBy** (*string \$group*)

Sets a LIMIT clause

abstract public *string* **getGroupBy** ()

Returns the GROUP BY clause

abstract public *string* **getPhql** ()

Returns a PHQL statement built based on the builder parameters

abstract public *Phalcon\ Mvc\ Model\ QueryInterface* **getQuery** ()

Returns the query built

## 2.54.298 Interface Phalcon\Mvc\Model\Query\StatusInterface

Phalcon\Mvc\Model\Query\StatusInterface initializer

### Methods

abstract public *Phalcon Mvc ModelInterface* **getModel** ()

Returns the model which executed the action

abstract public *Phalcon Mvc Model MessageInterface* [] **getMessages** ()

Returns the messages produced by a operation failed

abstract public *boolean* **success** ()

Allows to check if the executed operation was successful

## 2.54.299 Interface Phalcon\Mvc\Model\RelationInterface

Phalcon\Mvc\Model\RelationInterface initializer

### Methods

abstract public **setIntermediateRelation** (*string/array* \$intermediateFields, *string* \$intermediateModel, *string* \$intermediateReferencedFields)

Sets the intermediate model dat for has-\*through relations

abstract public *int* **getType** ()

Returns the relations type

abstract public *string* **getReferencedModel** ()

Returns the referenced model

abstract public *string/array* **getFields** ()

Returns the fields

abstract public *string/array* **getReferencedFields** ()

Returns the referenced fields

abstract public *string/array* **getOptions** ()

Returns the options

abstract public *string/array* **isForeignKey** ()

Check whether the relation act as a foreign key

abstract public *string/array* **getForeignKey** ()

Returns the foreign key configuration

abstract public *boolean* **isThrough** ()

Check whether the relation is a ‘many-to-many’ relation or not

abstract public *string/array* **getIntermediateFields** ()

Gets the intermediate fields for has-\*through relations

abstract public *string* **getIntermediateModel** ()

Gets the intermediate model for has-\*-through relations

abstract public *string/array* **getIntermediateReferencedFields** ()

Gets the intermediate referenced fields for has-\*through relations

### 2.54.300 Interface Phalcon\Mvc\Model\ResultInterface

Phalcon\Mvc\Model\ResultInterface initializer

#### Methods

abstract public **setDirtyState** (*boolean* \$dirtyState)

Sets the object's state

### 2.54.301 Interface Phalcon\Mvc\Model\ResultsetInterface

Phalcon\Mvc\Model\ResultsetInterface initializer

#### Methods

abstract public *int* **getType** ()

Returns the internal type of data retrieval that the resultset is using

abstract public *Phalcon\ Mvc\ ModelInterface* **getFirst** ()

Get first row in the resultset

abstract public *Phalcon\ Mvc\ ModelInterface* **getLast** ()

Get last row in the resultset

abstract public **setIsFresh** (*boolean* \$isFresh)

Set if the resultset is fresh or an old one cached

abstract public *boolean* **isFresh** ()

Tell if the resultset is fresh or an old one cached

abstract public *Phalcon\ Cache\ BackendInterface* **getCache** ()

Returns the associated cache for the resultset

abstract public *array* **toArray** ()

Returns a complete resultset as an array, if the resultset has a big number of rows it could consume more memory than currently it does.

### 2.54.302 Interface Phalcon\Mvc\Model\TransactionInterface

Phalcon\Mvc\Model\TransactionInterface initializer

## Methods

abstract public **setTransactionManager** (*Phalcon\ Mvc\ Model\ Transaction\ ManagerInterface* \$manager)  
Sets transaction manager related to the transaction

abstract public *boolean* **begin** ()  
Starts the transaction

abstract public *boolean* **commit** ()  
Commits the transaction

abstract public *boolean* **rollback** ([*string* \$rollbackMessage], [*Phalcon\ Mvc\ ModelInterface* \$rollbackRecord])  
Rollbacks the transaction

abstract public *string* **getConnection** ()  
Returns connection related to transaction

abstract public **setIsNewTransaction** (*boolean* \$isNew)  
Sets if is a reused transaction or new once

abstract public **setRollbackOnAbort** (*boolean* \$rollbackOnAbort)  
Sets flag to rollback on abort the HTTP connection

abstract public *boolean* **isManaged** ()  
Checks whether transaction is managed by a transaction manager

abstract public *array* **getMessages** ()  
Returns validations messages from last save try

abstract public *boolean* **isValid** ()  
Checks whether internal connection is under an active transaction

abstract public **setRollbackedRecord** (*Phalcon\ Mvc\ ModelInterface* \$record)  
Sets object which generates rollback action

### 2.54.303 Interface Phalcon\ Mvc\ Model\ Transaction\ ManagerInterface

Phalcon\ Mvc\ Model\ Transaction\ ManagerInterface initializer

## Methods

abstract public *boolean* **has** ()  
Checks whether manager has an active transaction

abstract public *Phalcon\ Mvc\ Model\ TransactionInterface* **get** ([*boolean* \$autoBegin])  
Returns a new Phalcon\ Mvc\ Model\ Transaction or an already created once

abstract public **rollbackPending** ()  
Rollbacks active transactions within the manager

abstract public **commit** ()

Commits active transactions within the manager

abstract public **rollback** ([*boolean* \$collect])

Rollbacks active transactions within the manager Collect will remove transaction from the manager

abstract public **notifyRollback** (*Phalcon\|Mvc\|Model\|TransactionInterface* \$transaction)

Notifies the manager about a rollbacked transaction

abstract public **notifyCommit** (*Phalcon\|Mvc\|Model\|TransactionInterface* \$transaction)

Notifies the manager about a committed transaction

abstract public **collectTransactions** ()

Remove all the transactions from the manager

### 2.54.304 Interface Phalcon\ Mvc\ Model\ ValidatorInterface

Phalcon\ Mvc\ Model\ ValidatorInterface initializer

#### Methods

abstract public *array* **getMessages** ()

Returns messages generated by the validator

abstract public *boolean* **validate** (*Phalcon\|Mvc\|ModelInterface* \$record)

Executes the validator

### 2.54.305 Interface Phalcon\ Mvc\ ModuleDefinitionInterface

Phalcon\ Mvc\ ModuleDefinitionInterface initializer

#### Methods

abstract public **registerAutoloaders** ()

Registers an autoloader related to the module

abstract public **registerServices** (*Phalcon\|DiInterface* \$dependencyInjector)

Registers services related to the module

### 2.54.306 Interface Phalcon\ Mvc\ RouterInterface

Phalcon\ Mvc\ RouterInterface initializer

## Methods

abstract public **setDefaultModule** (*string* \$moduleName)

Sets the name of the default module

abstract public **setDefaultController** (*string* \$controllerName)

Sets the default controller name

abstract public **setDefaultAction** (*string* \$actionName)

Sets the default action name

abstract public **setDefaults** (*array* \$defaults)

Sets an array of default paths

abstract public **handle** ([*string* \$uri])

Handles routing information received from the rewrite engine

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **add** (*string* \$pattern, [*string/array* \$paths], [*string* \$httpMethods])

Adds a route to the router on any HTTP method

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addGet** (*string* \$pattern, [*string/array* \$paths])

Adds a route to the router that only match if the HTTP method is GET

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addPost** (*string* \$pattern, [*string/array* \$paths])

Adds a route to the router that only match if the HTTP method is POST

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addPut** (*string* \$pattern, [*string/array* \$paths])

Adds a route to the router that only match if the HTTP method is PUT

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addDelete** (*string* \$pattern, [*string/array* \$paths])

Adds a route to the router that only match if the HTTP method is DELETE

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addOptions** (*string* \$pattern, [*string/array* \$paths])

Add a route to the router that only match if the HTTP method is OPTIONS

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addPatch** (*string* \$pattern, [*string/array* \$paths])

Add a route to the router that only match if the HTTP method is PATCH

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **addHead** (*string* \$pattern, [*string/array* \$paths])

Adds a route to the router that only match if the HTTP method is HEAD

abstract public **clear** ()

Removes all the defined routes

abstract public *string* **getModuleName** ()

Returns processed module name

abstract public *string* **getControllerName** ()

Returns processed controller name

abstract public *string* **getActionName** ()

Returns processed action name

abstract public *array* **getParams** ()  
Returns processed extra params

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **getMatchedRoute** ()  
Returns the route that matches the handled URI

abstract public *array* **getMatches** ()  
Return the sub expressions in the regular expression matched

abstract public *bool* **wasMatched** ()  
Check if the router matches any of the defined routes

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* [] **getRoutes** ()  
Return all the routes defined in the router

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **getRouteById** (*string* \$id)  
Returns a route object by its id

abstract public *Phalcon\ Mvc\ Router\ RouteInterface* **getRouteByName** (*string* \$name)  
Returns a route object by its name

abstract public *bool* **isExactControllerName** ()  
Returns whether controller name should not be mangled

### 2.54.307 Interface Phalcon\ Mvc\ Router\ RouteInterface

Phalcon\ Mvc\ Router\ RouteInterface initializer

#### Methods

abstract public *string* **compilePattern** (*string* \$pattern)  
Replaces placeholders from pattern returning a valid PCRE regular expression

abstract public **via** (*string/array* \$httpMethods)  
Set one or more HTTP methods that constraint the matching of the route

abstract public **reConfigure** (*string* \$pattern, [*array* \$paths])  
Reconfigure the route adding a new pattern and a set of paths

abstract public *string* **getName** ()  
Returns the route's name

abstract public **setName** (*string* \$name)  
Sets the route's name

abstract public **setHttpMethods** (*string/array* \$httpMethods)  
Sets a set of HTTP methods that constraint the matching of the route

abstract public *string* **getRouteId** ()  
Returns the route's id

abstract public *string* **getPattern** ()

Returns the route's pattern

abstract public *string* **getCompiledPattern** ()

Returns the route's pattern

abstract public *array* **getPaths** ()

Returns the paths

abstract public *string/array* **getHttpMethods** ()

Returns the HTTP methods that constraint matching the route

abstract public static **reset** ()

Resets the internal route id generator

### 2.54.308 Interface Phalcon\Mvc\UrlInterface

Phalcon\Mvc\UrlInterface initializer

#### Methods

abstract public **setBaseUri** (*string* \$baseUri)

Sets a prefix to all the urls generated

abstract public *string* **getBaseUri** ()

Returns the prefix for all the generated urls. By default /

abstract public **setbasePath** (*string* \$basePath)

Sets a base paths for all the generated paths

abstract public *string* **getbasePath** ()

Returns a base path

abstract public *string* **get** ([*string/array* \$uri], [*unknown* \$args], [*unknown* \$local])

Generates a URL

abstract public *string* **path** ([*string* \$path])

Generates a local path

### 2.54.309 Interface Phalcon\Mvc\ViewInterface

Phalcon\Mvc\ViewInterface initializer

#### Methods

abstract public **setViewsDir** (*string* \$viewsDir)

Sets views directory. Depending of your platform, always add a trailing slash or backslash

abstract public *string* **getViewsDir** ()

Gets views directory

abstract public **setLayoutsDir** (*string* \$layoutsDir)

Sets the layouts sub-directory. Must be a directory under the views directory. Depending of your platform, always add a trailing slash or backslash

abstract public *string* **getLayoutsDir** ()

Gets the current layouts sub-directory

abstract public **setPartialsDir** (*string* \$partialsDir)

Sets a partials sub-directory. Must be a directory under the views directory. Depending of your platform, always add a trailing slash or backslash

abstract public *string* **getPartialsDir** ()

Gets the current partials sub-directory

abstract public **setBasePath** (*string* \$basePath)

Sets base path. Depending of your platform, always add a trailing slash or backslash

abstract public *string* **getCurrentRenderLevel** ()

Gets the current render level

abstract public *string* **getRenderLevel** ()

Gets the render level for the view

abstract public **setRenderLevel** (*string* \$level)

Sets the render level for the view

abstract public **setMainView** (*string* \$viewPath)

Sets default view name. Must be a file without extension in the views directory

abstract public *string* **getMainView** ()

Returns the name of the main view

abstract public **setLayout** (*string* \$layout)

Change the layout to be used instead of using the name of the latest controller name

abstract public *string* **getLayout** ()

Returns the name of the main view

abstract public **setTemplateBefore** (*string/array* \$templateBefore)

Appends template before controller layout

abstract public **cleanTemplateBefore** ()

Resets any template before layouts

abstract public **setTemplateAfter** (*string/array* \$templateAfter)

Appends template after controller layout

abstract public **cleanTemplateAfter** ()

Resets any template before layouts

abstract public **setParamToView** (*string* \$key, *mixed* \$value)

Adds parameters to views (alias of setVar)

abstract public **setVar** (*string* \$key, *mixed* \$value)

Adds parameters to views

abstract public *array* **getParamsToView** ()

Returns parameters to views

abstract public *string* **getControllerName** ()

Gets the name of the controller rendered

abstract public *string* **getActionName** ()

Gets the name of the action rendered

abstract public *array* **getParams** ()

Gets extra parameters of the action rendered

abstract public **start** ()

Starts rendering process enabling the output buffering

abstract public **registerEngines** (*array* \$engines)

Register templating engines

abstract public **render** (*string* \$controllerName, *string* \$actionName, [*array* \$params])

Executes render process from dispatching data

abstract public **pick** (*string* \$renderView)

Choose a view different to render than last-controller/last-action

abstract public *string* **partial** (*string* \$partialPath)

Renders a partial view

abstract public **finish** ()

Finishes the render process by stopping the output buffering

abstract public *Phalcon\Cache\BackendInterface* **getCache** ()

Returns the cache instance used to cache

abstract public **cache** ([*boolean/array* \$options])

Cache the actual view render to certain level

abstract public **setContent** (*string* \$content)

Externally sets the view content

abstract public *string* **getContent** ()

Returns cached ouput from another view stage

abstract public *string* **getActiveRenderPath** ()

Returns the path of the view that is currently rendered

abstract public **disable** ()

Disables the auto-rendering process

abstract public **enable** ()

Enables the auto-rendering process

abstract public **reset** ()

Resets the view component to its factory default values

abstract public **bool** **isDisabled** ()

Whether the automatic rendering is disabled

### 2.54.310 Interface Phalcon\Mvc\View\EngineInterface

Phalcon\Mvc\View\EngineInterface initializer

#### Methods

abstract public **array** **getContent** ()

Returns cached output on another view stage

abstract public **string** **partial** (**string** \$partialPath)

Renders a partial inside another view

abstract public **render** (**string** \$path, **array** \$params, [**boolean** \$mustClean])

Renders a view using the template engine

### 2.54.311 Interface Phalcon\Paginator\AdapterInterface

Phalcon\Paginator\AdapterInterface initializer

#### Methods

abstract public **setcurrentPage** (**int** \$page)

Set the current page number

abstract public **stdClass** **getPaginate** ()

Returns a slice of the resultset to show in the pagination

### 2.54.312 Interface Phalcon\Session\AdapterInterface

Phalcon\Session\AdapterInterface initializer

#### Methods

abstract public **start** ()

Starts session, optionally using an adapter

abstract public **setOptions** (**array** \$options)

Sets session options

abstract public **array** **getOptions** ()

Get internal options

abstract public **mixed** **get** (**string** \$index, [**mixed** \$defaultValue])

Gets a session variable from an application context  
abstract public **set** (*string* \$index, *string* \$value)

Sets a session variable in an application context  
abstract public **has** (*string* \$index)

Check whether a session variable is set in an application context  
abstract public **remove** (*string* \$index)

Removes a session variable from an application context  
abstract public *string* **getId** ()

Returns active session id  
abstract public *boolean* **isStarted** ()

Check whether the session has been started  
abstract public *boolean* **destroy** ([*unknown* \$session\_id])

Destroys the active session

### 2.54.313 Interface Phalcon\Session\BagInterface

Phalcon\Session\BagInterface initializer

#### Methods

abstract public **initialize** ()  
Initializes the session bag. This method must not be called directly, the class calls it when its internal data is accessed

abstract public **destroy** ()  
Destroys the session bag

abstract public **set** (*string* \$property, *string* \$value)  
Setter of values

abstract public *mixed* **get** (*string* \$property, [*mixed* \$defaultValue])  
Getter of values

abstract public *boolean* **has** (*string* \$property)  
Isset property

abstract public **remove** (*string* \$property)  
Unset property

### 2.54.314 Interface Phalcon\Translate\AdapterInterface

Phalcon\Translate\AdapterInterface initializer

## Methods

abstract public *string* **query** (*string* \$index, [*array* \$placeholders])

Returns the translation related to the given key

abstract public *bool* **exists** (*string* \$index)

Check whether is defined a translation key in the internal array

## 2.54.315 Interface Phalcon\Validation\ValidatorInterface

Phalcon\Validation\ValidatorInterface initializer

## Methods

abstract public *mixed* **isSetOption** (*string* \$key)

Checks if an option is defined

abstract public *mixed* **getOption** (*string* \$key)

Returns an option in the validator's options Returns null if the option hasn't been set

abstract public **setOption** (*string* \$key, *mixed* \$value)

Sets the validator's option

abstract public *Phalcon\Validation\Message\Group* **validate** (*Phalcon\Validator* \$validator, *string* \$attribute)

Executes the validation

## 2.55 Лицензия

Phalcon создан в Phalcon Team! [[Twitter](#) - [Google Plus](#) - [Github](#)]

Фреймворк Phalcon PHP Framework распространяется под лицензией new BSD license. в случаях, когда не указано иное, содержимое текущего сайта распространяется под лицензией Creative Commons Attribution 3.0 License.

Если вам понравился Phalcon, поделитесь своей радостью и работами с сообществом! :)

## Документация в других форматах

---

- PDF
- HTML страницы в Zip
- ePub