Framework Foton

Версия 0.12.280

Документация для разработчиков

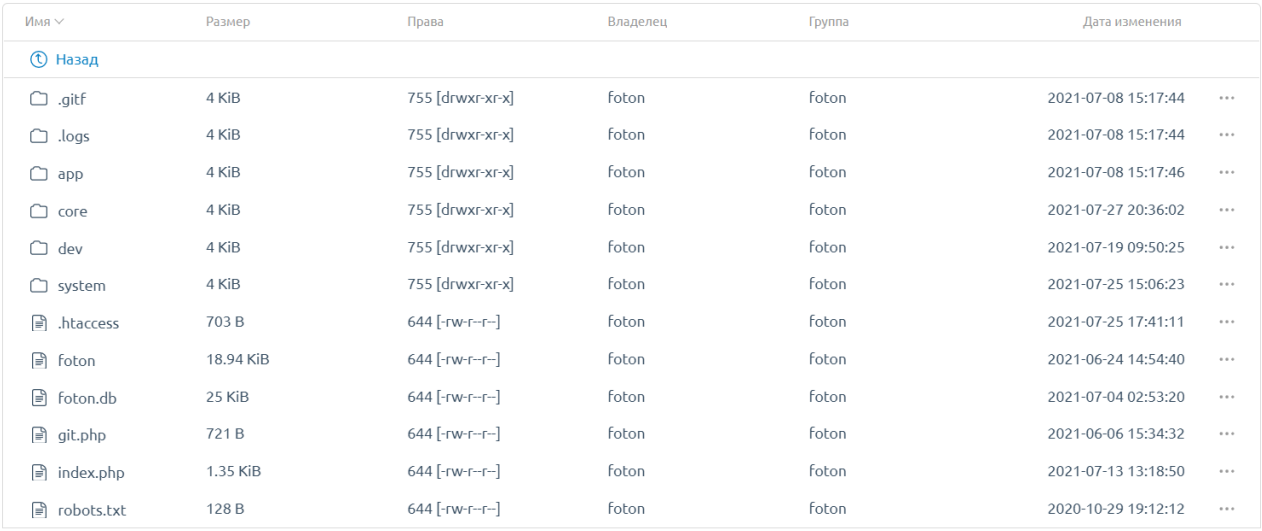
Требование к системе

- Сервер Apache 2.4+
- Тип сервера баз данных MySql/Postgresql/LiteSQL
- Версия интерпретатора php 5.6.* - 8.*.*
- ОС: Widows/Linux
- Установленные модули (zip,unzip)
- Библиотеки:
  1. Curl
  2. XMLWriter
  3. SoapClient (не обязательно)
  4. PDO
  5. Smtp
- Разрешенные функции
  1. exec
  2. eval

Содержание

14. Внутренние типы данных
15. Файловые типы данных
16. Интерфейсы
17. Интерфейс List
18. Шаблонизатор Foton
19. Composer Foton
20. Создание внутренних модулей системы
21. Создание внешних модулей системы
22. Работа с Ajax
23. Создание виджетов
24. Система интерфейсов
25. ЧПУ и оптимизация
26. Глобальный контроллер
27. Кастомизация и события
28. Проверка типов данных
29. Обработчик Handler
30. Обработчик Cron
31. Вертикальный шардинг
32. Работа с Memcached
33. Модульное тестирование
34. Консольное приложение Foton
35. Работа с транзакциями
36. Правила конвенции Foton
37. Зарезервированные элементы системы
38. Особенности системы

Структура системы



Имя	Размер	Права	Владелец	Группа	Дата изменения
Назад					
.gitf	4 KiB	755 [drwxr-xr-x]	foton	foton	2021-07-08 15:17:44
.logs	4 KiB	755 [drwxr-xr-x]	foton	foton	2021-07-08 15:17:44
app	4 KiB	755 [drwxr-xr-x]	foton	foton	2021-07-08 15:17:46
core	4 KiB	755 [drwxr-xr-x]	foton	foton	2021-07-27 20:36:02
dev	4 KiB	755 [drwxr-xr-x]	foton	foton	2021-07-19 09:50:25
system	4 KiB	755 [drwxr-xr-x]	foton	foton	2021-07-25 15:06:23
.htaccess	703 B	644 [-rw-r--r--]	foton	foton	2021-07-25 17:41:11
foton	18.94 KiB	644 [-rw-r--r--]	foton	foton	2021-06-24 14:54:40
foton.db	25 KiB	644 [-rw-r--r--]	foton	foton	2021-07-04 02:53:20
git.php	721 B	644 [-rw-r--r--]	foton	foton	2021-06-06 15:34:32
index.php	1.35 KiB	644 [-rw-r--r--]	foton	foton	2021-07-13 13:18:50
robots.txt	128 B	644 [-rw-r--r--]	foton	foton	2020-10-29 19:12:12

- Директория Gitf предназначена для хранения версий сайта у различных пользователей, подробнее в пункте 2.
- Директория .logs предназначена для хранения логов, подробнее в пункте 3.
- Директория app содержит само приложение и состоит из директорий: ajax,model,controller,view, а также необязательную директорию lang
- Директория app/ajax предназначена для хранения файлов контроллеров работающих при вызове ajax запросов. Состоит из файлов ajax_(site).php, ajax_(admin).php,ajax_(site)_m.php, ajax_(admin)_m.php. В названиях site и admin – названия текущих тем административной и публичной части приложения. Директории с такими названиями содержат (admin – php,js,css файлы интерфейсов,site – php,js,css файлы типов данных)
- app/controller – директория хранения контроллеров, внутри папка с названием темы сайта (по умолчанию site) и папка с названием темы панели (по умолчанию admin), общий контроллер админ. Панели admin_globals.php и общий контроллер сайта site_globals.php
- app/model – директория хранения моделей, внутри папка с названием темы сайта (по умолчанию site) и папка с названием темы панели (по умолчанию admin).

- app/view – директория хранения представлений, внутри папка с названием темы сайта (по умолчанию site) и папка с названием темы панели (по умолчанию admin), а также директория xml, директория json, и файл template.php, также может содержать foot.php и head.php.
- Core – ядро системы, содержит файлы model.php, core.php, config.php, view.php, core/lib/adapter.php, core/lib/lib.php, core/lib/preload.php, core/lib/run.php Все файлы обновляются при обновлении ядра, кроме config.php – в нем обновляется только версия ядра системы. Файлы core/lib/adapter.php и core/lib/lib.php отвечающие за рендеринг страницы и сбор данных можно переопределить в файле core/config.php.
- dev – директория для разработчика:
  - custom.php предназначен для модификации методов ядра.
  - event.php предназначен для обработки данных до или после выполнения методов ядра.
  Пример кода для переопределения метода test

```
public function before_Core_test($x,$y)
{
        return array($x+8,$y*2);
}
public function after_Core_test($x)
{
        return $x+20;
}
```

  - sharding.php предназначен для настройки вертикального шардинга данных для конкретной модели или интерфейса. Пример кода:

```
<?
$arr =array(
    "site"=>
    array("html.mvc"=>array(
        "dbname"=>'',
        "login"=>'',
        "pass"=>''
        )
    ),
```

```
    "admin"=>array(
        "interface([^/]+)/html"=>array(
        "dbname"=>'',
        "login"=>'',
        "pass"=>''
        )
    )
);
```

- type.php предназначен для определения типов данных методов ядра. Пример кода для метода ядра test:

```
<?
class Type{
 public function Core_test()
 {
        return ['array','string'];
      }
}
```

dev/modules – директория хранения внутренних модулей системы.
dev/modul – директория внешних модулей системы.
dev/widget — директория с виджетами системы, используются в основном для вывода определенных представлений на сайте, либо в типах данных.

- system – директория хранит системные данные, вход в панель управления, файлы шаблонов приложения, файлы шаблонизатора, а также шаблоны интерфейсов и сайтов для установки.
  - system/api/php директория — хранит шаблоны php кода для шаблонизатора Foton,шаблоны хранятся в директориях сайтов.
  - system/api/tpl директория — содержит файлы шаблонов представлений административной и публичной части приложения.
  - system/api/tpladmin директория — хранит директории тем интерфесов.
  - system/api/tplsite директория — хранит директории тем сайтов.
  - migrations директория для хранения миграций.
- Директория system/admin-inc предназначена для авторизации в

системе, для безопасности и совместимости ее не желательно изменять при разработке. Содержит страницу авторизации и выхода, скрипты и стили для страницы и события авторизации и выхода.

- Директория для хранения архивов сайта, название меняется в файле core/config.php, после этого переименуйте папку и можете делать архивы с помощью средств api системы, либо функционала интерфейса. Главное в названии директории использование спец. Символов, для того, чтобы директорию нельзя было считать посредством обращения чрез строку браузера.
- Git.php – файл для работы с внутренней системой gitf, подробнее в следующей главе.
- Foton – консольное приложение, список консольных команд в разделе «Консольное приложение».
- Index.php – основной индексный файл системы.
- .htaccess – файл apache. При использовании nginx замените на аналогичный, перенаправляет все запросы на индексный файл.
- Robots.txt – файл для поисковых систем.

Список таблиц системы

- graph — таблица для хранения графиков и вывода в стандартном интерфейсе
- buttonred — таблица хранения данных ExecCommand для типа данных «html текстовое поле»
- router — таблица для хранения чпу представлений (системная таблица)
- mapofsite — таблица для хранения представлений, которые выводятся в карте сайта (sitemap.xml)
- exfield – таблица хранения дополнительных полей (системная таблица)
- html — таблица хранения типов данных (системная таблица)
- mediateka — таблица хранения фотографий медиатеки (системная таблица)
- menu — таблица хранения меню
- seo — таблица хранения мета данных представлений (системная таблица)
- role — таблица хранения ролей пользователей (системная таблица)
- taxonomy — таблица для хранения типов таксономии

- user_inc — таблица хранения данных пользователей (системная таблица)

Установка системы

Для установки системы скачайте вашу версию с сайта foton.name и распакуйте архив в директорию вашего сайта.
Перейдите по адресу в строке браузера http{s}://вашсайт/install.php
Создайте базу данных и укажите ее параметры на данной странице, а также лицензионный ключ, после этого войдите в открытую форму указав везде demo, перейдите в модуль Обновление модулей, и обновите модули до последней версии, также рекомендуем обновить ядро системы в верхней панели, а также в разделе пользователи изменить ваши данные, более безопасно создать нового администратора, зайти под ним и удалить demo администратора, иначе после изменения логина вас выбросит из системы с ошибкой, и если вы не верно ввели пароль или забыли его вам придется вручную менять ваш пароль в базе данных. Также рекомендуем создать ваш личный ключ нажав на соответствующий значок в верхней панели РСИ, данный ключ требует синхронизации времени вашего устройства с вашим сервером.

Установка для LiteSQL - Для установки системы скачайте вашу версию с сайта foton.name и распакуйте архив в директорию вашего сайта.
Удалите лишние файлы pbase.sql,base.sql,install.php откройте файл core/config.php и укажите: $GLOBALS["foton_setting"]["host"]='localhost';
$GLOBALS["foton_setting"]["sql"]='lite';
$GLOBALS["foton_setting"]["dbname"]='foton';
Далее перейдите по адресу http{s}://вашсайт/admin/, далее шаги такие же как и для других SQL версий.

Система Gitf

Gitf – директория предназначена для хранения веток и рабочих директорий пользователей.
Имеет структуру -.gitf/login/work/ (рабочая директория)
.gitf/login/release/branch/ (ветка)
Для работы достаточно

1. Создать директорию внутри папки .gift по названию вашего логина, при установке уже создана директория пользователя demo.
2. Создать в этой директории директорию work или release, можно и то и другое.
3. Скопировать в директорию work файл который вы хотите изменить со всеми папками от корневой директории, например /view/site/head.php.
4. В случае создания ветки в директории release создаем папку с произвольным названием вашей ветки латинскими буквами и переносим необходимый к изменению файл как в пункте 3.
5. Переходим по адресу /git.php?test=yes, после этого для вас, как для авторизованного пользователя при работе с системой будет работать файл именно из вашей директории. При создании ветки, для обращения именно к этой ветке /git.php?test=yes&branch=название_ветки, тогда при обработке будет произведено обращение к файлу из этой ветки, если данный файл в ней существует.

Для поддержания функциональности таких событий при написании своих интерфейсов для работы с файловой структурой используйте метод $this->core->git($path), данный метод выполняет поиск файла в директории .gitf текущего пользователя.
Для работы с миграциями создайте также копию базы и разместите код с новыми доступами к базе данных для вашего пользователя <? $GLOBALS["foton_setting"]['dbname']='XXXXXX';$GLOBALS["foton_setting"]['login']='XXXXXX';$GLOBALS["foton_setting"]['pass']='XXXXXX';?> в файле .gitf/login/work/sql.php.

Работать с системой Gitf помогает модуль «Версии», с помощью этого модуля вы можете создавать рабочии директории, копировать базы данных

для пользователей с ролью 5 (разработчики).

Важно!!! Файлы ядра системы core/* и внешних модулей modul/* не инициализируются в Gitf системе.

Внутреннее логирование

При использовании интерфейса с сохранением post данных они сохраняются внутренней системой логирования в журнале /.logs – логи интерфейса. В данную директорию попадают только данные с name прописанным в глобальном массиве $GLOBALS["foton_setting"]['value_log']. По умолчанию это array('script','text');
логи хранятся по следующему пути - .logs/логин/число-месяц-год/_name(post)_часы_минуты_секунды.php

Авторизация по ключу

Авторизация в системе происходит не только по логину и паролю, но еще и по last code. Система использует 2 last code, которые задаются в файле system/admin-inc/key.php ($date,$date2) — переменные не шифруются и могут содержать любые символы и одинаковы для всех пользователей. Для безопасности настоятельно рекомендуется использовать генерацию ключей одного из стандартных интерфейсов. При генерации создается файл который генерирует 2 временные хеш метки, при открытии с помощью браузера, достаточно кликнуть в место хеш метки для выделения и копирования данных в буфер обмена, далее достаточно вставить метку в форму авторизации. Для работы хеш меток необходимо синхронизировать время на вашем компьютере с временем на сервере вашего сайта.

Ядро системы (роутер, рендер)

Ядро системы состоит из model.php, core.php, config.php, view.php
config.php — конфигурационный файл с глобальными переменными
core.php — само ядро, содержит методы для работы с системой, обращение в контроллерах и моделях $this->core->метод ядра.
View.php – является роутером и рендером одновременно. Здесь проверяется чпу, и выполняется подключение либо только представления без шаблона, либо результата метода, подключение контроллеров, моделей, выбор шаблона и представления.
View подключает файлы /core/lib/lib.php,/core/lib/adapter.php
Model.php – собирает все объекты системы для распределенного использования:
$this->core методы ядра
$this->glob методы глобального контроллера
$this->mod методы внутренних модулей
$this->widget подключение виджета
core/lib/preload.php – содержит родительский класс parent содержащий основные методы, в основном для работы с базой данных, также содержит класс validate содержащий основные методы валидации и классы widget и mod для работы с виджетами и внутренними модулями системы соответственно.

Системный рендеринг

Для работы с системным рендерингом добавьте настройку «render»=>true в ваш конфигурационный файл, после чего в контроллерах публичной части вашего приложения используйте только методы с названиями по следующим алгоритмам:
Для подключения метода на странице с форматом json добавьте метод json_page(),для xml – xml_page(), для обычных страниц mvc_page().
Для подключения метода с определенными get параметрами, например если у вас ЧПУ mvc /news/%href%:a-z:/%id%:0-9:.html mvc_href_id($href,$id),

для всех страниц просто href_id($href,$id), если вам нужно подключить только для определенного get параметра, допустим для вашего представления есть еще одно ЧПУ /news/%href%:a-z:/ используйте mvc_href($href) или просто href($href), если у вас есть 2 get параметра mvc и page, то метод будет работать с ними, а также еще раз подключится для вашего mvc_page(), поэтому не используйте get параметры page и ваших форматов вывода (json,xml,mvc,txt, etc...)
Get параметры в названии метода идут ровно в том порядке, в котором указаны в вашем ЧПУ.

Основные объекты системы

- $this->core методы ядра

пример: $this->core->globfunc('workarea');
работает в контроллере, модели, аякс контроллере **app/ajax/ajax_admin.php**, аякс модели **app/ajax/ajax_admin_m.php**, ajax интерфейсах **app/ajax/admin/php/интерфейс.php**, ajax типах данных **app/ajax/site/php/типданных.php**, контроллере, ajax контроллере и модели внешних модулей

- Методы моделей (для типов данных)

пример: $this->tpl_html('html')->htmlredactor('[[value]]','[[name]]');","2");
работает в таблицах типов данных

- $this->glob методы глобального контроллера

пример: $this->glob->timesc();
работает в контроллере, модели, аякс контроллере **app/ajax/ajax_admin.php**, аякс модели **app/ajax/ajax_admin_m.php**, ajax интерфейсах **app/ajax/admin/php/интерфейс.php**, ajax типах данных **app/ajax/site/php/типданных.php**, контроллере, ajax контроллере и модели внешних модулей

- $this->mod методы внутренних модулей

пример: $this->mod->названиемодуля->методмодуля();
работает в контроллере, модели, аякс контроллере **app/ajax/ajax_admin.php**, аякс модели **app/ajax/ajax_admin_m.php**, ajax интерфейсах

app/ajax/admin/php/интерфейс.php, ajax типах данных **app/ajax /site/php/типданных.php**, контроллере, ajax контроллере и модели внешних модулей

- $this->widget подключение виджета

пример: $this->widget->названиевиджета->методвиджета();

работает в контроллере, модели, аякс контроллере **app/ajax/ajax_admin.php**, аякс модели **app/ajax/ajax_admin_m.php**, ajax интерфейсах **app/ajax/admin/php/интерфейс.php**, ajax типах данных **app/ajax/site/php/типданных.php**, контроллере, ajax контроллере и модели внешних модулей

- $this->request – содержит массивы данных сессий ->s,Post данных ->p, Get данных ->g, массив данных сервера ->sr, массив request ->r
- Массив методов контроллера

$data['названиеметода'] – работает в представлении контроллера.

Как устроено внутреннее MVC?

Для создания mvc шаблона создайте контроллер в разделе controller/(admin|site|etc...)/controller_вашеимя.php

Внутри класс

```
<?
class Controller_Имя extends Model_Имя{

}
```

Как видим контроллер наследует модель

Далее создайте модель model/(admin|site|etc...)/model_вашеимя.php

```
<?
class Model_Имя extends Model{

}
```

Модель служит для хранения методов обработки готовых данных от контроллера (то есть get,post,request и т. д. Данных в ней не должно быть) Контроллер получает эти данные, и обрабатывает их проверяя на безопасность и корректность и передает в метод модели, так как модель родитель - $this->методмодели().

Методы контроллера не должны содержать сигнатур, при загрузке системы результаты всех методов контроллера записываются в массив $data, получить результат метода в представлении можно используя представление $data['методконтроллера'], а также $controller_class->метод(); $model_class->метод();

Методы ядра системы

git($GLOBALS["foton_setting"]["path"]."/dir/myfile.txt");

Ищет файл в системе gitf, если есть сессия gitf и находит файл возвращает путь из директории .gitf/, если нет возвращает текущий путь

Сигнатура: $path – строка, путь до файла $GLOBALS["foton_setting"] ["path"]."/dir/myfile.txt" или "dir/myfile.txt"

Результат: Путь до файла

fexists_foton('view/site','head.php');

Проверяет существование файла, проверяет сначала в директории текущей

админ. Панели, если нет то в директории текущего сайта, если и там файла нет проверяет напрямую в директории от корня сайта, можно использовать для проверки любого файла
Сигнатура: $file – название файла контроллера или модели полное (model_file.php или controller_file.php)
$mc – директория 'model' или 'controller'
Результат: Если файл существует возвращает путь до него, если нет возвращает false

tpl_html("myfile",1);
Подключает класс модели. Используется в основном для подключения методов моделей сайта в типах данных, отсюда и название.
Сигнатура: $tpl – краткое название модели, то есть если класс модели 'Model_MyCustom', то писать нужно только 'mycustom'
$dir 1 — текущая админ.
нет — текущий сайт
- указано другое, ищет в указанной директории
Результат:

Если находит, то подключает и возвращает объект класса, если нет, возвращает log с описанием ошибки.

arr_methods($tpl,$dir,$type) - arr_methods('workarea',1,2)
Возвращает массив списка методов класса
Сигнатура: $tpl – название mvc шаблона ('controller_workarea.php'=>'workarea')
$dir – 1 ищет в директории админки, null – ищет в директории сайта, другая строка — ищет в указанной директории
$type – 1 выводит методы модели, 2 — выводит методы контроллера, тип не указан выводит список методов уже подключенного класса.
Результат: Путь до файла

include_tpl($path) — include_tpl('menu/form_end');
Путь к файлу интерфейса из директории /ajax/admin/php/
Сигнатура: Название интерфейса/название файла

Результат: выводит путь к файлу, при ошибке возвращает log с описанием ошибки

globfunc ($mvc=null,$dir='cs') - globfunc ('workarea','ca')
Возвращает объект контроллера или модели
Сигнатура: $mvc – название mvc шаблона
('controller_workarea.php'=>'workarea')
$dir – cs контроллер/директория сайта, ms - модель/директория сайта, 'ma' - модель/директория админки, 'ca' — контроллер /директория админки
Результат: объект класса, в случае ошибки возвращает log

dir_search_foton ($path=null,$search=null,&$files=null) - dir_search_foton ($GLOBALS["foton_setting"]['path'].'/modules','word')
Возвращает массив ссылок на файлы, где было найдено совпадение по слову
Сигнатура: $path – путь к директории в которой нужно искать,
$search – фраза, которую нужно найти
$files – ссылка на массив в рекурсии, при запуске не передается.
Результат: массив путей к файлам, в которых было найдено совпадение.

translite_m ($text=null) - translite_m ('русский текст')
Выполняет транслитерацию строки
Сигнатура: $text — текст для транслитерации
Результат: Результат транслитерации — строка из латинских символов и знаков пунктуации.

translit_base ($text=null) - translit_base ('русский текст')
Выполняет транслитерацию строки
Сигнатура: $text — текст для транслитерации
Результат: Результат транслитерации — строка из латинских символов и знаков пунктуации.

del_file($table=null,$id=null) - del_file('table',5)
Удаляет файл по id в таблице. Путь к файлу может быть в любом столбце,

должен начинаться от корня текущего сайта без слеша, то есть если файл лежит в /view/site/img/file.txt – путь должен быть img/file.txt, несколько путей должны быть разделены %%%
Сигнатура: $table – таблица в базе данных
$id – id записи
Результат: Количество удаленных файлов, если не переданы таблица и id выводит log

dir_delete_foton($dir) -
Удаляет директорию
Сигнатура: $dir – полный путь до директории
Результат: При ошибке возвращает false

post_arr()
Возвращает массив post данных в массиве $post

Результат: массив post данных

post_get()
Превращает url вида a/b/c/d в массив $_GET[0]=a,$_GET[1]=b,$_GET[2]=c и т. д.

Результат: $_GET

include_files($file=null)
Выводит содержимое файла
Сигнатура: $file – путь от корня сайта со слешем в начале
Результат: содержимое файла

Методы работы в публичной части

dbins(string: $table,array: $arr,string: $model) - запись в таблицу $table
$model – название модели, необязательное значение, если указана модель в

методе перед обработкой вызываются хуки модели : $table."_validate" и $table."_is"
$arr = ['name'=>'value']

dbup(string: $table,array: $arr,array: $where,string: $model) – обновление данных в таблице
$model – название модели, необязательное значение, если указана модель в методе перед обработкой вызываются хуки модели : $table."_validate" и $table."_is"
$arr = ['name'=>'value']
$where = ['GL','=id'=>5,'AND','>count'=>7,'GR','OR','GL','%text'=>'test','AND','field'=>'test','GR'] В SQL запросе получим
WHERE (id=5 AND count>7) OR (text LIKE '%test%' AND field='test')

isvalid(array: $arr1,array: $arr2,string: $status) – проверяет данные на валидность
$arr1 = массив методов проверки, ключи поля значений
например: ['mail'=>'M','text'=>'O']
M – mail
D – домен
F – число float
IP – IP адрес
R – регулярное выражение
U – URL адрес
I – число integer
O – непустое значение
T – строка
$arr2 = ['mail'=>'mail@mail.ru','text'=>'my text']
$status – любая не пустая строка, если задана, то результатом будет значение true или false, если не указан результатом будет массив ключами которого будут поля, значениями true – поле введено верно, false – поле введено не верно.

validate(string | array: $arr1,array: $arr2) – изменяет данные в соответствии с

методами обработки
если $arr1 строка, то она должна содержать один из методов обработки, либо массив ключами которого будут ключи массива $arr2, значениями методы обработки.
Список методов обработки:
html — эквивалент htmlspecialchars
pass — создает md5 хеш
text — оставляет только печатные символы
medium — оставляет только русские и латинские символы и пробел
abcd — оставляет только русские и латинские символы, цифры и пробел
abc - оставляет только латинские символы
abv - оставляет только русские символы
same — оставляет данные без изменения
int – оставляет только цифры
phone — оставляет цифры ± и скобки
tags — удаляет все теги
plus — аналог метода abs
mini – аналог метода floor
maxi - аналог метода ceil
nul – всегда выдает null
tabs – удаляет все пробельные символы
$arr2 – массив полей со значениями
$arr2 = ['mail'=>'mail@mail.ru','text'=>'my text']

api(string: $key,string: $model,string: $table,array: $arr,integer: $id) – метод служит для изменения данных таблицы методами GET и POST
$key == GET['key']
$model == название модели
$table == название таблицы из модели
$arr – массив данных для обновления, добавления или вывода ['поле таблицы'=>'значение'], необязательный параметр, если массив $_POST == $arr.
Обязательные get параметры для запроса по api:

key – ключ приложения
status – может принимать несколько значений:
1. up – обновление данных
    обязательные параметры:
    $arr['id'] – id записи
2. I – insert запись данных
3. del – удаление записи:
    обязательные параметры:
    id – id записи
4. echo – вывод записи по id
    обязательные параметры:
    id – id записи
5. list – вывод нескольких записей
    не обязательные параметры
    page – страница,count – количество записей (используются только вместе)
    where – json строка массива $arr для выборки записей.
format - "api_json"

water_mark($image=null,$mark=null,$w=20,$h=20,$zoom=7) – выводит обработанную фотографию с водяным знаком.
$image – путь до картинки
$mark – путь до водяного знака
$w – ширина знака
$h – высота знака
$zoom – увеличение знака
Для элементов без фото можно установить глобальный параметр фото по умолчанию $GLOBALS["foton_setting"]['image_def']
$GLOBALS["foton_setting"]['water_mark'] — путь до водяного знака по умолчанию

search($text, $arr) – поиск похожего слова на слово $text из массива $arr

searchf($name, $list) – более точный поиск похожего слова на слово $name из массива $list

h_site() - адрес сайта с http{s}

p_site() - адрес страницы с http{s}

redirect_get($select=array(),$stop=10,$increment=1,$no_increment=array()) - выполняет редирект на текущую страницу с get параметрами для обхода цикла.
$select – массив, содержащий имена get параметров, к которым будет применена операция инкремента
$stop – ['id'=>5,'count'=>7] – при достижении любым гет параметром ближайшего значения цикл будет остановлен, также может быть числом, и тогда если любое из начения get параметров дойдет до него цикл будет остановлен.
$increment – число, на которое нужно увеличить каждый элемент, либо массив, содержащий имена get параметров в ключе и значения инкремента для них в значении. Также массив может содержать в значениях «eval:» - тогда будут работать шаблоны для get параметров, к примеру ['count'=>'eval:|id|*2'] – в каждом следующем запросе значение get параметра count будет в 2 раза больше get параметра id.
$no_increment – массив, содержащий имена get параметров, к которым не будет применена операция инкремента.

Пример:
Разместите в контроллере главной страницы следующий код:

```
<?
class Controller_html extends Model_html
{
    public function test(){
        if(isset($_GET['id']) && isset($_GET['count'])){
                $select = ['id','count','name'];
                $no_increment = ['name'];
```

```
            $stop = ['id'=>5,'count'=>20];
            $increment = ['id'=>1,'count'=>'eval:|id|*2'];
            $this->core->redirect_get($select,$stop,$increment,$no_increment);
        }
    }
}
```

Теперь перейдите по ссылке:
http://вашдомен/html/?id=1&count=1&name=test

nearest($arr, $arr_search, $count = 2) – выполняет поиск подходящего массива векторов $arr в массиве массивов векторов $arr_search, $count – длина массива для поиска, выводит позицию найденного элемента.

```
$arr1 = array(
        array(2,-2),
        array(-2,2),
        array(-7,8)
 ); - где ищем
$arr2 = array(-7,8); - что ищем
$count = 2  длина массива для поиска
nearest($arr1,$arr2,$count);
$arr1 = array(
        array(2,-2,7),
        array(2,-2,8),
        array(-7,8,8)
);
$arr2 = array(2,-2,7);
$arr = $this->core->nearest($arr1,$arr2,$count);
```

при $count=2 выдаст 1, при $count=3 выдаст 0, так как будет искать в 3-х мерном пространстве векторов.

smtpSend($mailTo, $subject, $message) – отправляет письмо через SMTP, требует указания параметра $GLOBALS["foton_setting"]['smtp'], в противном случае выдает лог ошибок.
$mailTo — E-mail получателя

$subject — заголовок письма
$message — тело письма

is_valid($val, $flag) — проверяет $val по флагу $flag на соответствие, $flag может принимать значение:
M – mail
D – домен
F – число float
IP – IP адрес
R – регулярное выражение
U – URL адрес
I – число integer
O – непустое значение
T – строка

Для валидации данных из формы в trait Custom_Validate файла dev/custom_valid.php пишите название вашего метода по названию контроллера, который орабатывает ваш запрос, например для контроллера Controller_html будет работать метод html , и указываете там массив для валидации полей в result, ключами будут поля, значениями методы валидации, например: return ['name'=>'text','date'=>'phone','age'=>'int'];
После этого отправляйте данные безопасно из вашей формы добавив поле foton_validate

Ни в коем случае не оставляйте название вашего метода в публичной части, так как этими данными могут воспользоваться.

import_csv($model,$table,$name,$path) – обновляет, либо добавляет данные в таблицу $table из csv файла.
$model – название модели таблицы, например html
$table – название таблицы
$name – имя [name] элемента загрузки файла в <input type="file" name="name">

$path – используется в методе i_upload, имя директории в которую будет загружен файл, если директории нет, она будет создана.
После загрузки файл автоматически удаляется.
Для добавления лучше использовать экспортированный с помощью export_csv файл. Данные обновляются по id записей в таблице, если id указан 0, то запись будет добавлена как новая. При импорте используется метод i_update.
В случае ошибки выводит лог системы.

export_csv($table,$where,$path) – метод экспортирует данные из таблицы в csv файл.
$table – таблица, из которой выгружаем записи.
$where – необязательный параметр, по умолчанию null, должен содержать массив ключ — маска и поле, значение — значение выборки, например ['!name'=>'test'], для выборки данных используется метод getlist, если указаны поля с префиксом поиска интерфейса find_, то они заменятся на %.

ORM Foton

step 1 ($tb — название таблицы)

create($tb=null) – создание таблицы
insert($tb=null) – запись в таблицу
table($tb=null,$f=null) – вывод из таблицы, $f (fields) – поля для вывода, если не указан выводятся все поля
d($tb=null) — удаление записи
trun($tb=null) — очистка таблицы
up($tb=null,$p=null) — обновление таблицы, $p строка запроса типа «поле='значение',поле='значение'»
drop($tb=null) – удаление таблицы

step 2 ($p – название поля)

where($w=null,$and=null,$group=null) выборка значений (условие если)
$w – строка типа «поле=значение» или массив array(”поле”=>”значение”)
ключ может содержать маску:

array("!id"=>5) – id!=5
array(">p"=>5,"<s"=>7) – p>5 и s <7
array("%text"=>'слово') – text содержит 'слово'
array("^text"=>'слово') – text начинается с 'слово'
array("$text"=>'слово') – text заканчивается 'слово'
array("!text"=>'слово') – text не содержит 'слово'
$and – и,или, по умолчанию «и»
$group – 1 вместо where выражение закрывается в скобки (поле1=2 and поле2='text'), используется только вместе с where(),orf(),andf()
2 - вместо where выражение остается без изменений, используется только вместе с where(),orf(),andf()
where() - добавляет в запрос WHERE
andf() - добавляет в запрос AND
orf() - добавляет в запрос OR

join($arr) – оператор JOIN (выполняет сравнение на соответствие условию между таблицами)
$arr – array('table1', //1 таблица
array('field1','field2'), // поля 1 таблицы поддерживает маски как в where-(!,<,>)
'table2', //2 таблица
array('field1','field2'), //поля 2 таблицы
'L'); //необязательное
I – inner join (по умолчанию)
L – left join
R – right join
O – outher join
C – cross join
F – full join
пример: array('role',array('id'),'user_inc',array('id'),'L');
return $this->core->table('role')->join($arr)->eq();

like($wp=null,$w=null) — более быстрый поиск like,
$wp – массив значений полей, либо строка с значением,
$w – массив полей, либо строка с полем

where_arr($wp=null,$w=null) - более быстрый поиск =,
$wp – массив значений полей, либо строка с значением,
$w – массив полей, либо строка с полем
insert_arr($arr=null) – массив значений для записи array('поле'=>'значение')
field($name,$value) – добавляет создание поля, работают все поля из настроек $GLOBALS["foton_setting"]['orm'][“драйвер orm*”]
*драйвер orm может быть lite,mysql,pgsql

key($p=null) — создает первичный ключ
limit($f=null,$limit=null,$prev=null) - сортировка DESC
$f – строка поле сортировки,
$limit – количество записей
$prev – номер записи с которой начать

limita($f=null,$limit=null,$prev=null) - сортировка ASC
$f – строка поле сортировки,
$limit – количество записей
$prev – номер записи с которой начать

lim($limit=null,$prev=null) сортировка DESC
$limit – количество записей
$prev – номер записи с которой начать

sorts($sp=null,$s=null)
$sp – поле сортировки
$s – сортировка asc или desc

one($p=null)
$p – поле сортировки, выводит одно значение, сортировка DESC
group($f=null)
группирует, $f – поле по которому произойдет группировка

Step 3

query() - выполняет запрос к базе данных
q() - выполняет запрос prepare – результат объект orm
forq($field_key,$field_value) - выполняет запрос результат массив значений, ключи поля таблицы, и номера, то есть структура вида
array(0=>array(0=>1,id=>1,1=>text,name=>text));
Аргументы метода являются необязательными,
$field_key — строка(поле таблицы)
$field_value — строка или массив(поле или поля таблицы)
если они указаны, то выводится массив с ключами значениями поля $field_key и значениями равными значениям поля $field_value, если $field_value является массивом, то значением поля $field_key будет массив значений.
eq() - выводит строку запроса (для тестирования)
cq($echo=null) — выполняет создание таблицы с полями в формате InnoDB, кодировка utf8
c($count=null,$param=array()) - выводит количество затронутых строк в запросе:
$count - если указана переменная не null будет выполнен метод fetchColumn(), если не указана или null rowCount()
$param - если $param = ['S'=>true] - экранирование " не будет выполнено, используется в основном для поиска в сериализованных или json данных

Методы интерфейсов

i_insert - ($c_name=null,$table=null,$arr=null,$prev=null,$after=null, $ins_id=null)
$c_name — название модели
$table — название таблицы
$arr – массив данных *array(поле=>значение,поле=>значение)*
$prev — метод модели обработки данных до выполнения
$after - метод модели обработки данных после выполнения
$ins_id – если не ноль возвращается id созданной записи, иначе выдает результат массив обработанных данных
Записывает данные в таблицу, выдает результат массив обработанных

данных или id записи, в случае неудачи выдает log

i_delete - ($c_name=null,$table=null,$id=null,$prev=null,$after=null)
$c_name — название модели
$table — название таблицы
$id – id записи в таблице
$prev — метод модели обработки данных до выполнения
$after - метод модели обработки данных после выполнения
выполняет удаление записи с id=$id, выдает результат $id, в случае неудачи выдает log

i_update - ($c_name=null,$table=null,$arr=null,$prev=null,$after=null)
$c_name — название модели
$table — название таблицы
$arr – массив данных array(поле=>значение,поле=>значение)
$prev — метод модели обработки данных до выполнения
$after - метод модели обработки данных после выполнения
Обновляет данные в таблице, выдает результат массив обработанных данных, в случае неудачи выдает log

i_create - ($c_name=null,$table=null,$dbs=null)
$c_name — название модели
$table — название таблицы
$dbs – формат таблицы (InnoDB,MySAM и т. д.)
Создает таблицу используя хук метод модели 'interfaces', если он не найден выполняет поиск 'interface_sp', в случае неудачи выдает log

i_alter - ($c_name=null,$table=null)
$c_name — название модели
$table — название таблицы
Удаляет и обновляет поля таблицы используя хук модели 'drop_interface', в случае неудачи выдает log

i_drop - ($c_name=null,$table=null)
$c_name — название модели
$table — название таблицы
Удаляет таблицу используя хук модели 'drop_interface', в случае неудачи выдает log

i_list_ajax - ($arr=null,$replace=null,$delete=null)

```
$arr=array('model'=>'html','table'=>'graph','interface'=>'sp','extra_arr'=>array('name'=>'lang'),'fields_table'=>'field','fields_type'=>'format_select');
$replace=array('wheres'=>'textarea');
$delete=array('tables');
```

i_arr - ($arr=null)

```
$arr=array('model'=>'model' //— модель
,'table'=>'table' // таблица
,'interface'=>'interface' //интерфейс
,'extra_arr'=>array() //дополнительные поля для вывода, поле в массиве таблицы хука модели->поле в типе данных
,'fields_table'=>'fields' //элемент таблицы хука модели содержащий поля таблицы
,'fields_type'=>'type'  //элемент таблицы хука модели содержащий типы данных
);
```

i- ($c_name=null)
$c_name — название модели
Результат — объект модели, в случае ошибки false

i_h1 - ($c_name=null,$table=null)
$c_name — название модели
$table — название таблицы

Выдает название модели используя 'names'

i_unlink - ($del=null,$arr=null)
Удаляет файлы, пути от корня директории представлений сайта без слеша, то есть вместо /view/site/img/img.jpg пишем img/img.jpg,
$del – если не null то функция выполняется, иначе выводит log
Выполнение функции: из массива $arr удаляются файлы, значениями массива могут быть строки типа img/img.jpg%%% img/img1.jpg%%% img/img2.jpg, тогда будут удалены все файлы в такой строке.
$arr array('img/img.jpg','img/img.jpg','img/img.jpg%%% img/img1.jpg%%% img/img2.jpg');
Если $arr ==null пути к файлам ищутся в массиве $_POST, выбираются все значения post массива и проверяются на существование файлов, если файлы существуют, они удаляются.

i_upload - ($up=null,$arr=null,$path_dir=null,$size=null,$format=null)
Загрузка файлов
$up - если не null то функция выполняется, иначе выводит log
$arr - массив для удаления файлов, как в предыдущем методе - array('img/img.jpg','img/img.jpg','img/img.jpg%%% img/img1.jpg%%% img/img2.jpg');
$path_dir – директория хранения файлов $GLOBALS["foton_setting"]['path'].'/view/'.$GLOBALS["foton_setting"]['sitedir'].'/'.$path_dir если не указан и существует директория img будет установлена директория img
$size – максимальный размер загружаемого файла, если не указан будет установлен размер $GLOBALS["foton_setting"]['size_file'] в Мб
$format – массив разрешенных к загрузке форматов файлов, если не указан будет установлен $GLOBALS["foton_setting"]['format']
В случае ошибки загрузки на каком-либо этапе выводится log

i_handler_ajax - ($data=null,$arr=null)

Обработчик данных для выполнения аякс запроса по выводу типов данных

```
$arr = array(
        'where'=>array( //условие для запроса
                'field'=>array( //массив полей и значений
                        %!=name'=>'value' //поля работают с маской как в orm Foton
                ),
                'and'=>'and|or'), //и или
                'count'=>int, //количество выводимых элементов
                'page'=>int, //номер страницы
                'sort'=>array( //сортировка поле->тип
                        'fields'=>'asc'
                ));
```

$data — либо массив такой же структуры, либо строка такой же структуры, только для разделения используются символы '|||' - массив 1 уровня, ':::' - массив 2 уровня

i_load_ajax - ($html=null,$data=null,$arr_new=null)
$html – содержимое логического представления. Например тега <polygon>, $data,$arr_new — массивы передаются в том же виде методу i_handler_ajax($data,$arr_new) для обработки sql запроса.

i_for_ajax($arr_type,$arr_field)
$arr_type – массив типов
$arr_field – массив полей
Выводит массив для вывода готовых типов [field|type]

i_front_css - ($i=null) $i -название интерфейса, выводит css интерфейса, если он существует по пути /ajax/'.$GLOBALS["foton_setting"]['admindir']./css/названиеинтерфейса.css

i_front_js - ($i=null) $i -название интерфейса, выводит js интерфейса, если он существует по пути /ajax/'.$GLOBALS["foton_setting"]['admindir']./js/названиеинтерфейса.js

i_list - ($face=null,$sql=null,$add_arr=null)
Основной метод интерфейсов для вывода данных
$add = array('replace'=>array('name'=>'test') — замена поля переменной (необязательный параметр)
,'delete'=>array('title') — удаление полей переменной из вывода (необязательный параметр)
,'add'=>array('field'=>array('name2'=>'test2') — добавление новых полей со значениями (необязательный параметр)
,'tpl'=>array('name2'=>'textarea') — тип данных для поля
,'extra'=>array('lang'=>'new_text') — дополнительный параметр интерфейса сопоставление ключа массива таблицы со значением поля [[fields]] у типа данных из таблицы html (необязательный параметр)
));
$sql = array('where'=>array('field'=>array('%name'=>'Фото') — фильтр вывода
,'and'=>'and') *— разделитель where sql запроса*
,'count'=>5 *— количество записей*
,'page'=>0 *— страница*
,'sort'=>array('id'=>'ASC')); *- сортировка*
$face=array('model'=>'html' *– модель*
,'table'=>'menu' *— таблица*
,'interface'=>'sp' *— интерфейс, можно не указывать, в данном случае берется хук модели interface_sp (select property)*
,'extra_arr'=>array('name'=>'lang') *– дополнительные поля ключ — поле в массиве таблицы хука модели, значение название поля для замены в типе данных [[fields]]*
,'fields_table'=>'field' *– ключ таблицы хука модели содержащий массив*

полей таблицы

,'fields_type'=>'format_select' *– ключ таблицы хука модели содержащий массив форматов вывода*

,'create'=>true *— создание или вывод, если true то создание и выводится только одна запись без вывода значений полей из таблицы базы данных*

,'create_value'=>array('field'=>'value') *– используется если 'create'=>true, пишет значения value полям field при выводе*

,'count'=>'pagination' *— поле в хуке модели содержащее количество элементов на странице*

);

i_arr_all - ($model=null,$table=null,$interface=null)

$model – имя модели, $table – название таблицы, если не указано, выдаст массив из всех таблиц

$interface — если не указан выводится interface_sp – select property если указан 1 выводится стандартный interfaces

Если указано иное выводится указанный интерфейс.

Если модель или метод не найдены выводит ошибку типа log

i_echo_type -($type=null,$arr=null)

Основной метод вывода типов данных

$type – указывается код типа из таблицы html

$arr = array('fields'=>array('name'=>поле таблицы,'value'=>значение),'model'=>модель (необязательный параметр),'table'=>таблица (необязательный параметр));

Методы обработки вывода

abc09($text) – удаляет из переменной все символы кроме латинских букв, цифр и нижнего подчеркивания, выводит обработанный текст

number_foton($text) – удаляет из переменной все символы кроме цифр,

выводит обработанный текст

text_foton($text) – удаляет из переменной все символы кроме букв и пробела, выводит обработанный текст

input_foton($text) – удаляет из переменной php и html теги, а также --, выводит обработанный текст

html_foton($text) – дважды декодирует html символы, выводит обработанный текст, используется для вывода типов данных

text_resizes_foton ($text=null,$n=null) — обрезает строку
$text – строка, которую нужно обрезать,
$n – количество символов, до которого нужно обрезать строку.

unsetPost($delete=null,$js=null) — выполняет редирект, если есть post данные
$delete — 1,
$js — не null редирект происходит на js, если null на php
field_table($table=null) – выводит массив полей таблицы $table

csrf($key) – используется для защиты от CSRF-атак, и устанавливается как значение скрытого поля в форме ввода данных, $key – метод валидации, по умолчанию pass, то есть вы можете просто использовать csrf()

validate_csrf($decrypt='pass',$request=null) – метод проверки произведена ли CSRF-атака, $decrypt должен соответствовать $key используемой при передаче в csrf (по умолчанию pass), $request – переменная из вашей формы, имеющая значение csrf($key)

Методы вывода данных

getXML($data) – преобразует массив в XML данные, и выводит их.
$data – массив данных любой вложенности.

getid($arr) - выводит поля из таблицы по id записи
$arr = array('table'=>таблица,'id'=>id записи для вывода,
'fields'=>поля для вывода, необязательный параметр,
'error'=> любое значение. Если ключ существует, и записей нет будет произведен редирект на 404 страницу,
'format'=> принимает значения:
'O' - (объект) выводит результат в виде объекта
'J' – (json) выводит json строку
'S' – (serialize) выводит сериализованную строку
'E' – (echo) выводит SQL строку запроса для тестирования
'X' – (xml) выводит результат в xml формате
Если ключ не задан выводит по умолчанию массив

getlist($arr) — выводит список значений полей таблицы, является оберткой handlersql
$arr – array('таблица',
'where' =>array('(маска)поле'=>'значение','(маска)поле'=>'значение'), *//если logic=>LOGIC значением будет строка типа: "поле1='значение' or поле2!=12" (маска) — пустое → равно, ! → не равно,%->LIKE,^ → начало строки,$ → конец строки, < → меньше,> → больше*
'fields'=>array('field1','field2') или 'field1,field2' поля для вывода, если не указан group, если group указан, поля для вывода заменяется на поля group
'group'=>array('field1','field2') или 'field1,field2' – поля для группировки
'logic', - LOGIC,AND,OR
'sort', - array('поле'=>ASC или DESC);
'count', - количество элементов
'page' — страница
'format' - принимает значения:
'O' - (object) выводит результат в виде объекта
'J' – (json) выводит json строку
'S' – (serialize) выводит сериализованную строку
'E' – (echo) выводит SQL строку запроса для тестирования
'X' – (xml) выводит результат в xml формате
Если ключ не задан выводит по умолчанию массив
)

handlersql($table=null,$arr=null,$format=null) — основной метод вывода данных для интерфейсов и пользовательских методов, основан на ORM Foton

$table – название таблицы

$arr - array('where'=>array(

'field'=>array('(маска)поле'=>'значение'),

// (маска) — пустое → равно, ! → не равно,%->LIKE,^ → начало строки,$ → конец строки, < → меньше,> → больше

'and'=>'and или or')

)

,'count'=>количество элементов,

'page'=>страница,

'sort'=>array('fields'=>'asc') *— сортировка, поле → ASC ,DESC,LOGIC, если LOGIC, то для ['where']['field'] значением будет строка типа:*

"поле1='значение' or поле2!=12");

$format - принимает значения:

'O' - (объект) выводит результат в виде объекта

'E' – (echo) выводит SQL строку запроса для тестирования

Если ключ не задан выводит по умолчанию массив

Методы работы с ролями и пользователями

chmod_id($arr) - $arr – массив id ролей, если роль текущего пользователя есть в массиве выводит true, если нет false, служит для проверки доступа текущего пользователя

isAuth() - проверяет авторизован ли пользователь в системе, если да, возвращает true, иначе false

chmod_section($table=null) – проверяет доступ текущего пользователя к таблице, если доступ есть возвращает true, иначе false

$table – таблица, для которой производится проверка доступа

user_ip() - выводит ip адрес пользователя.

Методы работы с внутренними модулями

m_method($path=null,$arg=null,$arr=null) подключает класс передает результат метода класса
$path — путь к файлу класса или фасад namespaces/method
$arr – необязательный аргумент может быть строкой или массивом, передается только как один аргумент для __construct класса при его инициализации.
о фасадах и классах более подробно в разделе «Разработка внутренних модулей»
если класса не существует пишет false
$arg – массив аргументов метода для функции call_user_func_array

m_class($path=null,$arr=null) — подключает класс, передает массив методов класса, ключ → название, значение → результат метода
$path — путь к файлу класса или фасад namespaces
$arr – необязательный аргумент может быть строкой или массивом, передается только как один аргумент для __construct класса при его инициализации.
о фасадах и классах более подробно в разделе «Разработка внутренних модулей»
если класса не существует пишет false

m_name($path=null) — подключает класс, передает название класса, используется, если контроллер вашего класса имеет более одного аргумента.
$path — путь к файлу класса или фасад namespaces
о фасадах и классах более подробно в разделе «Разработка внутренних модулей»
если класса не существует пишет false

Пример использования:
$name = $this->core->m_name('mytest');
$obj = new $name($arg1,$arg2,$arg3 etc.);

m_obj($path=null,$arr=null) - подключает класс,создает и передает объект класса
$path — путь к файлу класса или фасад namespaces
$arr – необязательный аргумент может быть строкой или массивом, передается только как один аргумент для __construct класса при его инициализации.
о фасадах и классах более подробно в разделе «Разработка внутренних модулей»
если класса не существует пишет false

Методы работы с типами данных

tpl_front_css($arr=null) — выводит содержимое css файла типа данных
$arr – массив содержащий названия типов данных, возможна передача вместе с разделителями и аргументами прямо из хука модели.

tpl_front_js($arr=null) — выводит содержимое js файла типа данных
$arr – массив содержащий названия типов данных, возможна передача вместе с разделителями и аргументами прямо из хука модели.

type_kod($f1=null,$f2=null) – выводит поле по типу данных из таблицы html
$f1 – название типа данных
$f2 – название поля для вывода, если не указано будет передан массив со всеми полями

extracod($cod=null) – выводит значение дополнительного поля по его коду

imgcod($cod=null) – выводит путь к картинке от корня сайта по коду в медиатеке

Методы отладки

is_format($format=null,$get=0) — проверяет формат в строке ЧПУ.
$format - .xml,.json,.modul и т. д.
$get – если указан не ноль, проверяется только точное совпадение
Возвращает true или false
arr() Печать массива
Результат: Печать массива <pre> print_r() </pre>

log($text=null,$format=null,$method=null) - log('текст',1,'number_foton')
Выводит сообщение в консоль
Сигнатура: $text – текст для вывода, может быть массивом, $format – если не null массив сериализуется перед выводом, $method – перед выводом переменная $text – обрабатывается методом, если он указан, методы должны принадлежать ядру core и иметь одну переменную сигнатуры
Результат: вывод console.log

log_file ($name='no',$text=null,$append='no')
Создает файл в директории .logi
$name если не равна no файл создается с этим именем и суффиксом Ymd.!?
Иначе Ymd.!?
$text – текст для записи обязательный параметр
$append – если не no, то данные дозаписываются в файл

alert($text=null,$method=null,$format=null)
Выводит js сообщение alert
$text – текст для вывода
$method – если не null, данные обрабатываются этим методом, это только метод ядра, который имеет только один первый обязательный параметр
$format — если не null данные перед выводом сериализуются

st_start() - запускает запись времени выполнения скрипта

st_end() - останавливает запись времени и выводит результат в секундах
eq() - выводит строку запроса (из ORM Foton)

Методы работы с базой данных

select_db ($table=null,$attr=null,$attr2=null,$echo=null)
$table — таблица
$attr — поле таблицы в условии WHERE
$attr2 — значение поля таблицы в условии WHERE
$echo — поля для вывода через ; или одно поле
Выводит массив данных значений полей таблицы

update_db ($table=null,$attr=null,$attr2=null,$where=null)
$table — таблица
$attr — поля для обновления через ;
$attr2 — значения полей через ;
$where — строка условия WHERE

delete_db($table=null,$attr=null,$attr2=null)
$table — таблица
$attr — поле таблицы в условии WHERE
$attr2 — значение поля таблицы в условии WHERE
Удаляет записи из таблицы удовлетворяющие условию

field_tables($table=null)
$table — таблица
Выводит массив полей таблицы

field_table_foton($table=null,$field=null)
$table — таблица
$field – поля для вывода
Выводит массив значений поле таблицы

strukture_foton

format_table_foton($table=null)
$table — таблица
Выводит массив форматов полей таблицы

id_field_foton ($table=null,$field=null,$id=null)
$table — таблица
$field – поля для вывода
$id – id записи для вывода в условии WHERE
Выводит массив значений полей таблицы

insert_db($table=null,$attr=null,$attr2=null)
$table — таблица
$attr — поля таблицы для записи через ;
$attr2 — значения полей таблицы через ;
Создает новую запись в таблице

table_listdesc()
Выводит массив таблиц текущей базы данных, в desc содержится массив полей таблиц
в id содержится массив названий таблиц

site_dump($dump=null,$arr_no_del=null)
$dump – если не null то выполняется запись текущей базы данных в файл, иначе ошибка и сообщение log
$arr_no_del – массив таблиц, которые не нужно записывать в файл
Создается архив файловой системы и базы данных в директорию $GLOBALS["foton_setting"]['key']

sql_insert($file=null) — выполняет запись в базу данных из файла
$file – путь к файлу дампа базы данных
tables_sp_foton() - выдает массив списка таблиц текущей базы данных

sql_dump_file($file=null,$no_table=null) — создает дамп базы данных и

сохраняет в файл
$file – абсолютный путь к файлу для сохраниения дампа
$no_table – массив названий таблиц, которые не нужно записывать в дамп (необязательный параметр)

select_db_seo($table=null,$attr=null,$attr2=null,$echo=null)
$table — таблица
$attr — поле таблицы в условии WHERE
$attr2 — значение поля таблицы в условии WHERE
$echo — поля для вывода через ; или одно поле
Выводит массив данных значений полей таблицы, аналогичен select_db, но имеет особенности для работы с таблицей seo

select_from_seo($table=null,$from=null,$to=null,$echo=null)
$table — таблица
$from - от какой записи выводить
$to — количество записей для вывода
$echo — поле сортировки desc
Выводит массив записей таблицы

tablessortw($valtables=null,$id="id",$sort='DESC',$ceil='0',$w1=null,$w2=null)
Выводит данные из таблицы
$valtables — название таблицы
$id – поле сортировки
$sort — тип сортировки
$ceil — если 0, сортируется как текст, если 1 как цифры
$w1 – поле проверки
$w2 – проверяемое значение для поля

Основные методы системы

update_core() - устанавливает соединение с ядром системы, получает версию ядра

up_core_conn() - передает массив для выполнения запроса по обновлению ядра

up_core($yes=null) — обновляет ядро системы, если $yes!=null

cache_foton($view=null) – преобразует шаблоны системы в php код,
a_cache_foton($view=null) – преобразует шаблоны системы в php код для представлений админ. панели,
$view – код шаблона для обработки
Метод выдает обработанный код
error404() - выводит 404 ошибку с заголовком и выводом шаблона $GLOBALS["foton_setting"]["error404"]

list_model($dir=null) – выводит массив с названиями моделей указанных в хуке names
$dir – директория содержащая модели в директории model (site,admin etc...)

s_mvc($path=null,$name=null,$dirw=null)
$path – view | model | controller
$name – название шаблона
$dirw — директория шаблона внутри view | model | controller (site,admin etc...)
Метод выводит содержимое элемента шаблона view | model | controller

mobile_foton() - определяет мобильное устройство, если устройство определено как мобильное возвращает true, иначе false

mail_foton($from=null,$to=null,$subject=null,$message=null,$file=null)
Отправляет E-mail со вложением
$from – Email от кого
$to – Email кому
$subject — тема письма
$message – текст письма, возможно в html формате, кодировка utf-8
$file – необязательный параметр, абсолютны йпуть к файлу на сервере

meta() - обрабатывает ЧПУ, и определяет мета данные страницы, результатом является массив мета данных с ключами названия мета данных: keywords,title,description

list_files_glob() - возвращает массив названий всех mvc шаблонов системы используя хук модели nameinclude

list_model_glob ($model=null)
$model – название модели
Возвращает массив таблиц данной модели доступных для редактирования текущим пользователем

list_mvc($dirs=null,$dirview=null)
$dirs — директория mvc (controller|model|view)
$dirview – директория сайта или админ. Панели
Выводит массив названий mvc шаблонов в формате
$arr[название файла модели:директория сайта или админ панели] = название модели из хука nameinclude

isset_file ($file=null) — проверяет есть ли модель или контроллер в директории сайта или админ панели в gitf системе, если да, возвращает true, иначе false

copy_dir($source=null, $dest=null, $over=1) — копирует файлы из одной директории в другую
$source — абсолютный путь к директории откуда копируем
$dest — абсолютный путь к директории куда копируем

sizefile($dir=null) — считает общий объем данных php файлов лежащих по пути $dir от корня сайта в байтах, считает без вложенных директорий

sizefiles($dir=null) — считает общий объем всех файлов лежащих по пути $dir в байтах, считает с вложенными директориями

require_class($mod=null,$dir=null) — служит для подключения модели и

контроллера, работает только если и модель и контроллер у шаблона mvc созданы, возвращает класс модели.
$mod – название модели, $dir – директория внутри директории контроллера и модели

require_obj($mod=null,$dir=null) — служит для подключения модели и контроллера, работает только если и модель и контроллер у шаблона mvc созданы, возвращает объект модели.
$mod – название модели, $dir – директория внутри директории контроллера и модели

delete_post($arr=null,$field=null)
удаляет данные из массива
$arr — массив (ключ->значение)
$field — массив, либо одно значение ключа для удаления
возвращается обработанный массив, если одно из значений не указано выводится log

Методы сортировки

arr_sort($arr=null,$sort=1)
$arr – массив для сортировки
$sort – если 1, массив сортируется по возрастанию значений, иначе по убыванию значений

arr_shift($arr_func=null,$item=null) — сдвигает массив на количество элементов $item
$arr_func – исходный массив
$item – число на которое нужно сдвинуть массив вперед
если $item='ARRAY' будет выполнен перебор всех вариантов сдвига массива
Результат — массив, который сдвинут на n элементов, либо массив всех переборов сдвигов массива

arr_sort_key($arr=null,$num=null,$sort=null)
Сортирует массив по длине ключей
$arr - массив, содержащий буквенные ключи
$num - максимальное количество ключей одинаковой длины
$sort - если не пусто, то массив сортируется по длине ключа от большего, иначе от меньшего к большему
Результат — массив отсортированный по длине ключей

Текучий интерфейс var

Методы текучего интерфейса var

var_start($var=null) – начало интерфейса, передаем строку или массив в метод для выполнения операций.
var_inarr($arr=null) — выполняет поиск в переданном массиве строки из интерфейса (возвращает true/false).
var_inarr_to($str=null) — выполняет поиск переданной строки в массиве интерфейса (возвращает true/false).
var_im($d='') - превращает массив интерфейса в строку по переданному разделителю $d.
var_ex($d=null) - превращает строку интерфейса в массив по переданному разделителю $d.
var_method($method=null) — выполняет переданный метод со строкой или массивом интерфейса.
var_replace($pattern,$replace) — заменяет $pattern на $replace используя str_replace.
var_preplace($pattern,$replace) — заменяет $pattern на $replace используя preg_replace.
var_search($search) – ищет в строке интерфейса переданную строку и записывает в строку интерфейса результат strripos.
var_search_to($search)– ищет в переданной строке строку интерфейса и записывает в строку интерфейса результат strripos.
var_input($path) — записывает строку интерфейса в файл по пути $path, если в интерфейсе массив, записывает его json обьект.

var_output($path) — сохраняет в строку интерфейса данные из файла по пути $path.
var_end() - выводит строку или массив интерфейса и обнуляет его.
var_count() - выводит количество символов в строке, если массив, то количество элементов массива.

Каскадные объекты

dump_obj($arr,$obj='O') – превращает массив в многоуровневые объекты. $arr- массив, $obj – флаг. По умолчанию работает как обертка метода mask_to_obj. Пример структуры массива:

```
$arr = array(
    '::number_foton#test'=>
        array('v'=>
            array('test123--'),
            'm'=>array(
            '::input_foton'=>array(
                                'v'=>array(
                                    'number_foton_test'=>'1'
                                )
                            )
                )
            )
    );
print_r($this->core->dump_obj($arr));
```

результат:

```
stdClass Object
(
    [number_foton_test] => stdClass Object
        (
            [input_foton] => 123
        )

)
```

Разберем массив - ::number_foton#test — так как в начале ::, то будет

выполнен поиск метода ядра, если бы не было :: была бы вызвана функция php библиотеки. Например мы могли бы написать str_replace, если бы мы указали, например Controller_Test::test – был бы подключен метод класса Controller_Test, класс должен быть динамическим и объявленным. #текстбезпробелов – это комментарий к методу, если будут объявлены два одинаковых метода, то результаты перезапишутся, чтобы этого не произошло мы добавляем префикс. Значением этого ключа, как и всех других в этом многомерном массиве является массив из двух ключей — v массив сигнатур метода, m – массив из новых методов, который состоит также из ключа — метода и значения массива v=>array(),m=>array(0 и т. д. Для использования результата предыдущего метода, как значения сигнатуры следующего в массиве v в ключе необходимо указать название метода заменив # на _, а также, если есть класс метода, то и его указываем в начале, вот так вот, например Controller_Test_test, в значении укажем любое значение, обычно 1.

Значение результата метода изменяются каскадно, поэтому их можно использовать несколько раз.

Например такой массив:

```
$arr = array(
    '::input_foton#test'=>
        array('v'=>
            array('test123<>--'),
            'm'=>array(
                'strlen'=>array(
                                'v'=>array(
                                    'input_foton_test'=>'1'
                                )
                            ),
            '::number_foton'=>array(
                                'v'=>array(
                                    'input_foton_test'=>'1'
                                )
                            )
                )
            )
```

```
    );
```

выдаст после обработки:

```
(
    [input_foton_test] => stdClass Object
        (
            [strlen] =>9
            [number_foton] => 123
        )

)
```

Как мы видим strlen = 9, это значит, что взят результат метода «test123<>».

Файловый интерфейс

В отличии от обычных интерфейсов данный интерфейс имеет жестко заданные параметры начальных полей json объекта.
Задаются параметры свойством Ядра Main arr_property_file и содержат массив: array('TYPE', 'VALUE', 'NAME', 'ID')
Данный интерфейс реализован к обычный mvc шаблон и реализован только в одном из стандартных РСИ «SECTION», но вы можете самостоятельно создать свой интерфейс для работы с пользователями скопировав вышеуказанный mvc ifiles со стилями и скриптами, либо создать свой воспользовавшись методами ядра для работы с ним:

Важно!!! Интерфейс требует наличия глобальной переменной, указывающей где будут хранится файлы интерфейса, по умолчанию в core/config.php это
$GLOBALS["foton_setting"]['ifile'] = $GLOBALS["foton_setting"]['path'].'/app/controller/'.$GLOBALS["foton_setting"]['admindir'].'/file';
Название справочника может содержать любые символы разрешенные в именовании файлов вашей ОС.
Для корректной работы mvc ifiles в РСИ «SECTION» необходимо наличие хотя бы одного справочника.

ifile_create($file) – метод создает файл по вышеуказанному пути в формате

$file.'.txt' и заполняет его начальной структурой, метод ничего не возвращает.

ifile_start($file) — присваивает значение аргумента статической переменной ifile, метод ничего не возвращает.
$file – абсолютный путь к справочнику.
ifile_update($arr) – обновляет запись
$arr – массив содержащий поля структуры и значения, ID – содержит id элемента для его обновления, например:
$arr = array('TYPE'=>'html', 'VALUE'=>'test value', 'NAME'=>'code', 'ID'=>2);
В случае успеха возвращает true, в случае ошибок выводит лог ошибок log

ifile_insert($arr) — создает запись
$arr – массив содержащий поля структуры и значения, ID – содержит id элемента содержит count('ID') так как инкремента в данной системе не предусмотрено, его нужно получить самостоятельно на стороне интерфейса, например:
$arr = array('TYPE'=>'html', 'VALUE'=>'test value', 'NAME'=>'code', 'ID'=>3);
В случае успеха возвращает id записи, которую создали, должен совпадать с вашим переданным id, в случае ошибок выводит лог ошибок log.

ifile_delete($id) – удаляет запись по ее id
$id – элемент в структуре ['ID']
В случае успеха возвращает true, в случае ошибок выводит лог ошибок log.

ifile_drop() - удаляет справочник,
если не найдена статическая переменная $ifile выдает лог ошибки, если файл успешно удален возвращает true, при ошибки удаления возвращает false.

ifile_arr() - если не найдена статическая переменная $ifile возвращает false, если найдена возвращает массив справочника.

ifile_list($default_name='Значение',$select = array('TYPE','NAME','VALUE')) –

выводит обработанный методом ядра f_echo_type тип элемента.
Если не найдена статическая переменная $ifile выдает лог ошибки.

icode($code,$file) – возвращает значение элемента файлового интерфейса по его коду.
$code – название кода элемента
$file – имя справочника(файла)
Пример:
$test = $this->core->icode('test','file');
Переменная при хорошем исходе будет содержать значение элемента справочника.

Внимание!!! Если несколько элементов этого справочника с одинаковыми кодами, будет возвращено значение первого найденного

В случае ошибок выводит лог ошибок log.

Работа с различными языками

Если ваше приложение должно работать на нескольких языках создайте в директории app директорию lang.
Для использования различных языков в модулях создайте файл:
/app/lang/modul/вашязык/вашмодуль/index.php
Для главного файла модуля и
/app/lang/modul/вашязык/вашмодуль/{item}.php
Для остальных файлов модуля, если таковые имеются, {item} – название файла лежащего в директории item вашего модуля

Для создания языкового файла для представления mvc вашего приложения создайте файл /app/lang/{dir}/вашязык/{страница}
{dir} – директория одного из сайтов или РСИ,в которой лежит ваше представление, например site
{страница} – название вашего представления, пример:

представление html_view.php – файл html_view.php
представление html_view.tpl – файл html_view.php
создайте в этом файле переменные и используйте в своем представлении.
Для смены языка измените в core/config.php глобальный параметр $GLOBALS["foton_setting"]['lang'] на имя вашей языковой директории.
Так как директория не является обязательной, для установки интерфейса или шаблона сайта в систему в которой нет языковой директории ее необходимо создать:
/app/lang/{dir} -для сайта или РСИ

Интерфейс List Select Property(sp)

Хук интерфейса Select Property (sp) представляет собой метод класса interface_sp
В котором содержится массив для вывода

```
$arr = array(
"таблица"=> array("field" => array("id","views","tables","sections"),
//поля таблицы
"format" => array("int","text","text","text"),
//формат данных хранения в базе данных столбцов
'text'=>'text(64300) NOT NULL',
'int'=>'int(70) NOT NULL',
'date'=>'DATE NOT NULL',
'bit'=>'BIT NOT NULL',
'poly'=>'POLYGON NOT NULL',
'real'=>'REAL NOT NULL',
'time'=>'TIME NOT NULL',
'medium'=>'mediumtext NOT NULL'
"name" => array("id","Представление","Таблица","Раздел"),
//extra code название полей для вывода
"format_select" => array("||ids||","||input||","||input||","||textarea||"),
//типы данных для вывода элемента
"format_select_list" => array("||ids||","||text||","||text||"),
//типы данных для вывода списком
```

```
"field_filter"=>array("id","views","tables"),
//поля для фильтра поиска
"text_filter"=>array("id","Представление","Таблица"),
//описания для фильтра поиска
"format_filter"=>array("||ids||","||input||","||input||"),
//формат типа данных для вывода в поиске
"format_sort"=>array("||ids||","||submit||","||submit||"),
//формат типа данных для вывода сортировки
"pagination"=>"10" //количество элементов в списке
,"key" => "id" //ключ таблицы бд
),
```

Миграции и хуки модели

```
public function nameinclude(){
    return 'описание'; //описание модели

}
public function names(){
    $arr = array('таблица'=>'описание'); //массив таблиц модели с описанием
    return $arr;

}
public function (таблица)_chmod(){
    return '1,1'; //id ролей, которым доступна таблица
}
public function before_ins_(таблица)($arr=false){
if(isset($mass)){
$arr... //преобразовываем массив данных таблицы перед созданием записи
return  $arr;
}
public function before_up_(таблица)($arr=false){
if(isset($mass)){
$arr... //преобразовываем массив данных таблицы перед обновлением записи
```

```
return  $arr;
}
public function after_ins_(таблица)($arr=false){
if(isset($mass)){
$arr... //преобразовываем массив данных таблицы после создания записи
return  $arr;
}
public function after_up_(таблица)($arr=false){
if(isset($mass)){
$arr... //преобразовываем массив данных таблицы после обновления записи
return  $arr;
}
public function before_del_(таблица)($id){
if(isset($mass)){
$arr... //преобразовываем массив данных таблицы перед удалением записи
return  $arr;
}
public function after_del_(таблица)($arr=false){
if(isset($mass)){
$arr... //преобразовываем массив данных таблицы после удаления записи
return  $arr;
}
public function before_select_(таблица)($value,$name){
 if(isset($value) && $name=='поле для изменения'){
        $value =$new_value; // изменяем значение конкретного поля перед выводом, хук берется в цикле данных, так что можно через elseif или case проверить и изменить каждое значение

    }
    return $value;
}
//метод исправляет данные при добавлении и обновлении
//при использовании методом валидации нескольких параметров передавайте в значение элемента массива массив, первым значением которого будет название метода, остальными его значения, например так 'public_ins'=>['name'=>['max','20']
```

```
public function (таблица)_validate(){
$arr = [
'public_ins'=>['name'=>'tags'] //при добавлении с публичной части приложения
'ins'=>['name'=>'tags'] //при добавлении с административной части приложения
'up'=>['name'=>'tags'] //при обновлении с административной части приложения
'public_up' =>['name'=>'tags'] //при обновлении с публичной части приложения
'delete'=>['public_ins'=>[],'public_up'=>[]] //поля для удаления
];
return $arr;
}
//метод проверяет данные при добавлении и обновлении
//при использовании методом валидации нескольких параметров передавайте в значение элемента массива массив, первым значением которого будет название метода, остальными его значения, например так 'public_ins'=>['name'=>['max','20']
public function (таблица)_is(){
$arr = [
'public_ins'=>['name'=>'tags'] //при добавлении с публичной части приложения
'ins'=>['name'=>'tags'] //при добавлении с административной части приложения
'up'=>['name'=>'tags'] //при обновлении с административной части приложения
'public_up' =>['name'=>'tags'] //при обновлении с публичной части приложения

];
return $arr;
}

public function drop_interface(){
    $arr = array(
"таблица" => array(
"field" => array('field1','field2') //столбцы удаляются из таблицы
),
"таблица" => array(
    "del" => "0" // если =>1 то таблица удаляется, если 0, либо данного массива нет таблица не меняется
    )
);
```

```
return $arr;

}

public function alter_interface(){
    $arr = array(
"таблица" => array(
'alter'=>array('field1','field2'), //столбец field1 переименовываем в столбец field2
'type'=>array('field2'=>'text') //задаем для нового столбца формат text, доступны форматы —
'text'=>'text(64300) NOT NULL',
'int'=>'int(70) NOT NULL',
'date'=>'DATE NOT NULL',
'bit'=>'BIT NOT NULL',
'poly'=>'POLYGON NOT NULL',
'real'=>'REAL NOT NULL',
'time'=>'TIME NOT NULL',
'medium'=>'mediumtext NOT NULL'
   );
return $arr;
   //не забудьте сменить поле во всех интерфейсах модели, иначе создастся новое поле, а старое не удалится, для добавления нового поля просто добавьте его в массив интерфейса вместе с форматами и другими ключами ваших интерфейсов
}
public function interfaces(){
   $arr = array(
"(таблица)" => array(
"field" => array('id','name','text'), //поля таблицы в базе
"name" => array('id','Номер группы','Название группы'), //описание полей, ключ для интерфейса sp или list, может отсутствовать
"format" => array('int','text','text'), //формат полей в базе данных
"format_select" => array("||ids||","||number||","||input||"), //формат типов данных
"key" => "id" //ключ таблицы
));
return $arr;
```

```
}
```

Логические представления (Polygon)

Вывод нескольких элементов на странице:

```
<? $arr=array('%login'=>'d');?>
<polygon table='user_inc' where="<?=serialize($arr);?>" model='users'
interface="1" controller="users" page='0' count='10' sort_f='name' sort='asc'>
[name|textarea]
[login|html]
</polygon>
```

Атрибуты тега polygon:

model=название модели

table = таблица для вывода

interface="1" если 1, будет вызван хук interface, если указать например sp, будет вызван интерфейс interface_sp

controller=название контроллера

'where'=*//условие для запроса*

сериализованный массив array(*//массив полей и значений*

%!=name'=>'value' *//поля работают с маской как в orm Foton*

),

'count'=int, *//количество выводимых элементов*

'page'=int, *//номер страницы*

'sort'='asc|desc'

'sort_f'='поле сортировки'

Вывод формы создания элемента:

```
<? $value=array('name'=>'test','login'=>'test2');?>
<polygon table='user_inc' model='users' interface="1" controller="users"
value_create='<?=serialize($value);?>' create='1'>
[name|textarea]
```

[login|html]
</polygon>
Атрибуты тега polygon:
model=название модели
interface="1" если 1, будет вызван хук interface, если указать например sp, будет вызван интерфейс interface_sp
controller=название контроллера
'create'=1 или любому значению, если параметр есть, то вместо вывода списка выводится один элемент с пустыми значениями для создания элемента таблицы
Можно использовать вместе с value_create
value_create =array('поле'=>'значение поля')
Тогда при найденных значениях они будут подставляться в такое поле.

Внутренние типы данных

Типы данных находятся в таблице html, и являются основой для вывода пользовательских элементов в интерфейсах. Типы данных относятся к данным сайта, так как выводят их и являются неотъемлемой частью самого сайта, независимо от интерфейсов.
Тип данных состоит из полей:
название — поле name,
название латинское – поле kod,
html код – поле html,
аргументы -поле argument,
php код – поле function
При указании в поле html '2'
мы выбираем второй тип, и в поле function мы можем указать php код для вывода без указания тегов php кода <?,?>
Также можно использовать для вывода данных методы модели, например методы модели html, таким образом они также перенесутся при копировании шаблона сайта. Используя код, например: return $this->tpl_html('html')->mehod(аргументы), подключится метод method модели html, с указанными аргументами. В самом методу модели можно вызывать виджеты с аргументами, таким образом можно обращаться к любым

методам любых моделей сайта. Вызывать напрямую виджеты или глобальные методы не получится, так как типы данных подключаются из ядра, где еще не определены объекты виджетов и глобального контроллера.
В аргументах нужно указать аргументы и переменные, которые используются в коде, указываются названия без знака $ через запятую.
Также в коде используют [[name]] — имя поля таблицы при выводе, [[value]] — значение поля при выводе.
Если указать html код в поле html, а в поле аргумент и php код (функция) указать 0, то тип будет выводится как html код.
[[lang]]<input type='mail' name='[[name]]' value='[[value]]'>, так как это html код, то обычные переменные и аргументы здесь уже не работают. Стоит заметить, что поля с типом html работают быстрее при выводе, в отличии от полей с php кодом.
[[lang]] — в данном случае пользовательский шорт-код, если его нет в интерфейсе этот код просто удалится, создавать можно любой шорт-код, только если на стороне интерфейса его интерпретируют. Для корректной работы с другими интерфейсами пишите правильно css,js код типа Данных, чтобы при отсутствии значения пользователю не отображался пустой непонятный блок и не появлялись js ошибки в консоли.
Любой тип Данных может содержать css,js,и php код для ajax запросов. О выводе данных вам не нужно заботится, об этом позаботится интерфейс, вам только нужно правильно расположить свои данные.
В директории ajax корня сайта есть директория по названию подключенного сайта
(название в файле /core/config.php $GLOBALS["foton_setting"]['sitedir']),
в этой директории и лежат типы данных вашего сайта. Напомним, что типы данных относятся именно к сайту, не к админ. Панели, и копируются при создании шаблона сайта, но не при создании интерфейса.
В этой директории находятся папки css,js,php для хранения css,js,php файлов соответственно.
Если мы только что создали новый тип данных у него должно быть уникальное в вашей таблице название на латинском (поле kod)
этим названием и называем файл в директории css будет нашкод.css, в директории js нашкод.js, в директории php нашкод.php.
Соответственно css файл будет подключен при подключении вашего типа

данных, js будет подключен при подключении вашего типа данных, php должен содержать php класс

```
<?
class Tpl_вашкод extends Model{

}?>
```

Внутри этого класса давайте создадим метод mymethod

```
<?
class Tpl_вашкод extends Model{
public function mymethod(){
        echo 'hello World';
}
}?>
```

Теперь при обращении /вашкод/mymethod.tpl на экране появится 'hello World'.
Вы можете через js код вашего типа данных отправлять любые данные через ajax на ваши методы контроллера этого типа данных, и обрабатывать их на стороне контроллера и возвращать результат пользователю. Таким образом типы данных позволяют полностью перенести работу по обработке стандартных полей на плечи сайта, а интерфейсам дать возможность создавать сами интерфейсы для обработки результатов данных и работы с хуками моделей сайта.

Файловые типы данных

При разработке CRM системы, или сложной панели управления сайтом может понадобится использование специальных типов данных независимо от выбранного шаблона сайта. Для этого можно воспользоваться файловыми типами данных. Файлы таких типов данных должны быть расположены в директории app/ajax/admin/type/. Для создания типа данных

с кодом input необходимо создать файл input.php и input.tpl, input.php отвечает за логику, не обязательный, должен содержать класс:

```
<?
class Type_input{
    function __construct(){
        global $core;
        $this->core = $core;
    }
    public function index($field){
        return $field;
    }
}
```

Файл index.tpl содержит вывод html кода, в нем доступна переменная $input(по названию поля), данная переменная — это результат метода index и может иметь любой тип, будь то строка, массив или объект и т. д.
Файловые типы также как и обычные могут иметь css и js код, для этого просто добавьте app/ajax/admin/type/css/input.css, app/ajax/admin/type/js/input.js, при этом для ajax запросов обработка происходит в классах app/ajax/admin/type/php все по тем же правилам, что и для обычных типов данных в app/ajax/site/php
Для подключения файловых типов данных необходимо добавить необязательный параметр 'type'=>'file'.

Интерфейсы

Интерфейсы являются mvc шаблонами административной панели.
Для создания интерфейса необходимо в названии файла и класса указывать ключевую фразу Interface, например:
создаем файл controller_interface_list.php

```
<?
class Controller_Interface_list extends Model_Interface_list{
```

```
}
?>
```

создаем файл model_interface_list.php

```
<?
class Model_Interface_list extends Model{
}
?>
```

Создаем файл interface_list_view.php

Размещаем файлы как и обычный шаблон в директориях соответственно:

/controller/admin/

/model/admin/

/view/admin/

В интерфейсах помимо стандартных хуков типа dir(), используются также хуки:

```
<? parent(){
return 'sp';
}
parent_method(){
return array('ready','js_i','css_i');
}
?>
```

Таким образом будут инициализированы методы ready,js_i,css_i контроллера Controller_Interface_sp, контроллер Controller_Interface_sp должен находится в той же директории, что и дочерний.

Интерфейсы имеют возможность создания собственных css,js стилей и php файлов для обработки ajax запросов.

Для создания css стиля интерфейса создайте в директории /ajax/

$GLOBALS["foton_setting"]['admindir']/css/

файл с названием вашего шаблона, например list.css

Для создания js скрипта интерфейса создайте в директории /ajax/

$GLOBALS["foton_setting"]['admindir']/js/
файл с названием вашего шаблона, например list.js
Для создания php класса интерфейса для ajax обращений создайте в директории /ajax/$GLOBALS["foton_setting"]['admindir']/php/
файл с названием вашего шаблона, например list.php
со следующим содержимым:

```
<?
Class Interface_list extends Model{

}?>
```

Добавьте свой метод, например test

```
<?
Class Interface_list extends Model{
public function test(){
return 'Hello World';
}
}?>
```

Тогда перейдя по url **list/test.face** вы увидите на экране
Hello World
Интерфейсы, как и другие контроллеры имеют хук метод dir(), в нем указывается в какой директории /view/ искать представление для контроллера, например:

```
  public function dir(){
        return 'admin';
  }
```

а также для отключения регионов шапки и футера:

```
  public function region(){
        return false;
```

```
}
```

Метод не обязательный, если директория не указана, представление берется из директории контроллера, то есть если контроллер лежит в папке /controller/site/, то и представление будет инициализировано из /view/site/.

Интерфейс List

_construct() - создает статические переменные:
self::$extra_arr - массив сравнений дополнительных полей интерфейса
self::$extra_find - массив сравнений дополнительных полей для поиска
self::$interface — название основного интерфейса
self::$model — название модели
self::$table — таблица
self::$i_def — реальный интерфейс, который подключается в контроллере, после проверки наличия в методе модели
self::$interface_arr — массив подключенного интерфейса
Методы интерфейса
page() - номер текущей страницы
js_i() - получаем js интерфейса и типов данных
css_i() - получаем css интерфейса и типов данных
h1() - метод выводит название раздела
ready() - метод загрузки инициализирует создание, удаление и обновление полей таблицы и выдает результат для проверки (работает с миграциями)
callback() - пользовательский метод инициализирует запись, удаление и обновление данных таблицы (работает с данными текущей таблицы)
find_value() - пользовательский метод выводит массив значений для фильтра поиска в формате поле=>значение
find_sql() - обертка предыдущего метода для передачи параметров в метод обертки ORM
ascdesc() - определяет текущее поле сортировки данных таблицы
sort_field() - определяет текущую сортировку данных таблицы
filter_find() - выводит фильтр поиска
filter_sort() - выводит поля для сортировки

echo_lists() - выводит данные из таблицы в списке
create() - выводит форму для создания нового поля таблицы
update() - выводит форму для обновления полей таблицы
is_list() - проверяет есть ли интерфейс в модели
list_one() - если интерфейс не найден выводит стандартный интерфейс модели для редактирования данных.

Шаблонизатор Foton

Шаблонизатор Foton предназначен для упрощения чтения и редактирования кода представлений.
При использовании шаблонизатора код вида

```
<?foreach($arr as $value){
if(isset($value['name']){?>
<a href="<?=$value['href'];?>" id="id<?=$value['id'];?>>
<?=$value['name'];?></a>
<?}}?>
```

Приобретет вид

```
{{news_1head}}
<a href='{{news_2href}}' id="id{{news_3id}}">{{news_4name}}</a>
{{news_5end}}
```

Код намного короче, и проще читается.
Это лишь пример, название news или 2href и т. д. Вы можете называть как угодно. В этом состоит главное отличие от шаблонизаторов типа smarty, также шаблоны не работают через интерпретатор, а хранятся в директории api/tpl/ и работают лишь для сохранения данных через интерпретатор, таким образом реальная страница выглядит также как в первом варианте и грузится быстрее чем при использовании интерпретатора.
Для создания метода шаблона можно воспользоваться модулем php шаблоны, либо создать директорию с уникальным названием внутри

api/php/$GLOBALS["foton_setting"]['sitedir'], внутри этой директории создаются файлы расширением php с любым php кодом, для интерпретации этого кода достаточно вызвать {{названиедиректории_названиефайла}}
Название директории и файла не должно содержать знака «_».
С версии 0.5.0 появились новые синтаксические конструкции:
@if{...} == <?if(...){?>
@if[...] == <?if(...){?>
@else@ == <?}else{?>
@elsif{...} == <?}else if(...){?>
@for{...} == <?for(...){?>
@for:{...} == <?foreach(...){?>
@elsif[...] == <?}else if(...){?>
@for[...] == <?for(...){?>
@for:[...] == <?foreach(...){?>
@arr{...} == <?print_r(...);?>
@csrf{} или @csrf{'вашметод'} - <input type="hidden" name="csrf" class="csrf" value="результат csrf метода">
@:@ == <?}?>
@::@ == <?}}?>
@:::@ == <?}}}?>
@{a} == <?=$a;?>
@[a] == <?=$a;?>
@tpl(...) == include указанного шаблона без _view из /system/api/tpl/
@tpl(... , ...) == include указанного шаблона без _view из /system/api/tpl/
(1 аргумент директория сайта в /system/api/tpl/ , второй название шаблона)
@rc(...) == require_once исполняемого php представления приложения из /app/view/[sitedir]/, в названии указывается полное название файла без .php
@inc(...) == include исполняемого php представления приложения из /app/view/[sitedir]/, в названии указывается название файла без _view.php
@inc(... , ...) == include исполняемого php представления приложения из /app/view/[sitedir]/, в названии указывается название файла без _view.php
(1 аргумент директория сайта в /app/view/, второй название файла)
внутри {} не должно быть фигурных скобок. Как и внутри [] недолжно быть квадратных скобок, именно по этой причине 2 формы записи.
Внутри inc(),rc(),tpl() не должно быть кавычек, названия файлов должны

писаться так @inc(site,html).

Composer Foton

Composer Foton является элементом внутреннего модуля FotonSystem и предназначен для загрузки и выгрузки внутренних модулей системы. Для работы с ним достаточно воспользоваться командой composer в командной строке: php foton composer module:create, где module название модуля для установки. Основным файлом настройки Composer является файл /dev/modules/FotonSystem/config.json

```
{
    "module":
    {
        "sdek": //название модуля для установки
            ["dpd","pochta"] //зависимые методы, устанавливаются вместе с ним
    },
    "php":
    {
        "sdek":
            ["php5-mysql","php5-curl"] //библиотеки php необходимые для работы модуля
    },
    "ignore":
    {
        "dpd":true //модули, которые не должны обновляться
    },
    "host":
    {
        "sdek":
        {
            "url":"https://foton.name/module.ajaxsite","key":"key_license" // адрес откуда должен быть обновлен модуль, и ключ авторизации
        }
    },
    "key":"your_key" //ключ авторизации вашего приложения, может быть
```

любой строкой, при установке данного ключа в предыдущем элементе вместо key_licrnse и замене url на адрес вашего сайта https://вашсайт/module.ajaxsite будет добавлена возможность получения модулей вашей системы из другой системы
}
Также для работы ваших обновлений на стороннем сайте необходимо разместить метод module в /app/ajax/ajax_site.php из файла /dev/modules/FotonSystem/readme.txt

Создание внутренних модулей системы

Внутренние модули хранятся в директории /modules/
Обращение к внутренним модулям происходит посредством объекта системы $this->mod->названиемодуля->методмодуля();
Для создания модуля необходимо создать файл по названию вашего модуля с расширением php в корне указанной ранее директории и объявить в нем класс по названию вашего модуля, например:
mymodule.php

```
<?
class mymodule{
	public function test(){
		return 'test';
	}
}
?>
```

Тогда обращение будет выглядеть вот так:
$this->mod->mymodule->test();
Так как часто при создании большого модуля бывает необходимо обращение к другим модулям, либо разбиение на несколько классов система предусматривает создание такой сложной структуры.

Хорошим тоном считается именно такое создание внутреннего модуля: Файл mymodule.php из предыдущего примера является контроллером, хотя и может принимать аргументы методов, в отличии от контроллеров mvc Foton, где каждый метод должен отрабатывать и без аргументов, так как рендеринг предполагает запись результатов всех методов контроллера в массив data.

Методы этого контроллера в связи с теми или иными условиями, вызванными пользователем или же самой системой обращаются к методам других классов этого или же другого модуля.

Для этого создаем директорию по названию нашего модуля на одном уровне с нашим файлом модуля (такая практика является хорошим тоном, так как облегчает поиски при анализе вашего модуля).

Mymodule/

class1.php

```
<?
namespace mymodule;
class class1{
 function __construct(){
        $this->core = new ₩Core;
    }
}
?>
```

class2.php

```
<?
namespace mymodule;
class class2{

}
?>
```

Мы создали 2 файла, как видим названия классов также соответствуют названию файлов.

Первый класс как мы видим наследуется от ядра, давайте теперь создадим внутри директории mymodule директорию custom/ → mymodule/custom/class1.php
Как мы видим название данного файла и соответственно класс совпадают, но благодаря namespace мы можем обращаться к обоим класса в одном файле.

```
<?
namespace mymodule₩custom;
class class1{

}
?>
```

Для упрощения работы с модулями используется 3 основных метода ядра, благодаря им нам не нужно подключать файлы и писать use, это также является хорошим тоном, так как упрощает поиски методов класса.

---

m_method($path=null,$arg=null) - подключает класс передает результат метода класса
$path — путь к файлу класса или фасад namespaces/method
о фасадах и классах более подробно в разделе «Разработка внутренних модулей»
если класса не существует пишет false
$arg – массив аргументов метода для функции call_user_func_array

m_class($path=null) — подключает класс, передает массив методов класса, ключ → название, значение → результат метода
$path — путь к файлу класса или фасад namespaces
о фасадах и классах более подробно в разделе «Разработка внутренних модулей»
если класса не существует пишет false

m_obj($path=null) - подключает класс,создает и передает объект класса

$path — путь к файлу класса или фасад namespaces

о фасадах и классах более подробно в разделе «Разработка внутренних модулей»

если класса не существует пишет false

Теперь свяжем все классы между собой

mymodule.php

```
<?

class mymodule{
 function __construct(){
        $this->core = new ₩Core;
    }
        public function test($class,$return=null){

                if($class==1){
                $class1 = $this->core->m_obj('mymodule/class1');
                return $class1->test($return);
                }
        else if($class==2){
                $class2 = $this->core->m_obj('mymodule/class2');
                return $class2->test();
        }
else{
$class1 = $this->m_obj('mymodule/class1');
                return $class1->test();

}
}
}
?>
```

Mymodule/

class1.php

```
<?
namespace mymodule;
class class1{
 function __construct(){
        $this->core = new ₩Core;
    }
    public function test($return=null){
        if($return==null){
            return 'Это класс class1 метод test  Пустое значение вывода';
        }
        else{
            $class1 = $this->m_obj('mymodule/custom/class1');
                return $class1->test($return);
        }

    }
}

?>
```

class2.php

```
<?
namespace mymodule;
class class2{
    public function test(){
        return 'Это класс class2 метод test';
    }
}
?>
```

custom/class1.php

```
<?
namespace mymodule₩custom;
class class1{
 public function test($return){
 return 'Это класс custom/class1 метод test переданное значение '.$return;
 }
}
?>
```

Для упрощения namespace большой вложенности можно использовать фасады, для этого достаточно поместить в файл /modules/вашмодуль/facade.ini (если этого файла нет, вы можете создать его самостоятельно)
строчку, например:
custom=mymodule/custom/class1
После этого в файле mymodule/class1.php
можно писать вместо
$class1 = $this->m_obj('mymodule/custom/class1');
просто
$class1 = $this->m_obj('mymodule/custom');
Теперь вызвав
$this->mod->mymodule->test(1,'привет');
мы получим 'Это класс custom/class1 метод test переданное значение привет'

Внимание!!! В момент инициализации внутренних модулей инициализировано только ядро системы, попытка обращения к другим объектам системы приведет к ошибкам.

Создание внешних модулей системы

Внешние модули отличаются тем, что могут иметь рабочий интерфейс для взаимодействия с пользователем.
Внешние модули хранятся в директории /modul
Структура внешнего модуля
Внешний модуль хранится в директории с названием модуля.

Назовем модуль module

Создадим папку

/modul/module

Структура папки

css – папка со стилями модуля, все файлы в корне этой директории подключаются автоматически при инициализации модуля

js - папка со скриптами модуля, все файлы в корне этой директории подключаются автоматически при инициализации модуля

item – директория содержит страницы модуля

module_a.php — ajax методы модуля

module_c.php — контроллер модуля

module_m.php — модель модуля

index.php — главный файл модуля

install._! - файл инициализации, если он есть, значит модуль активен, если нет, значит модуль выключен, либо удален в зависимости от настроек модели модуля.

module_a.php — содержит одноименный класс, по названию файла и наследуется от контроллера.

```
<?
class Module_a extends Module_c{

}?>
```

Создадим метод method1

```
<?
class Module_a extends Module_c{
public function method1(){
echo 'Метод 1';
}
}?>
```

Теперь при выполнении аякс запроса, например в файле /modul/module/js/script.js

```
 $.ajax({
            type: "POST",
            url: "/module/method1.ajax",
             success: function(data){
                console.log(data);
             }
        });
```

Или при переходе в браузере на страницу /module/method1.ajax
мы получим - Метод 1

Внимание!!! Метод, как и Модуль инициализируется только для авторизованных пользователей с определенными правами.

module_c.php — содержит одноименный класс, по названию файла и наследуется от модели.

```
<?
class Module_c extends Module_m{

}?>
```

Создадим метод post1

```
<?
class Module_c extends Module_m{
public function post1(){
if(isset($_POST['num'])){
return  $this->number($_POST['num']);
}
}
}?>
```

module_m.php — содержит одноименный класс, по названию файла и наследуется от модели ядра.

```
<?
class Module_m extends Model{

}?>
```

Создадим метод number

```
<?
class Module_m extends Model{
public function number($text=null){
if(i$text!=null){
return   $this->core->number_foton($text);
}
else{
return $this->core->log('Ошибка в   Module_m->number не указана сигнатура');
}
}?>
```

Теперь при передаче данных POST **$_POST['num']** например из файла index.php в переменной **$data['post1']** будет содержаться числовое значение преданного post параметра.

Методы модели модуля:

```
public function name_m(){
            return 'Мой модуль'; //метод передает название модуля

        }
        public function modul_chmod(){
    return '1,2'; //метод передает список ролей пользователей, для которых данный модуль будет активным
```

```
}
public function version(){
return '1'; //версия вашего модуля целое или дробное число, например 1.5
}
 public function version_up(){
return array("host"=>"(ваш url)","date"=>array(массив данных при обновлении));
}

public function install(){
$connect=array("index"=>array("host"=>"(ваш
url)","method"=>"post","date"=>array(массив данных при инсталляции))); //
записываем данные ответа в переменную $data["index"]
);
return $connect;
}
```

после инициализации install был произведен запрос на наш url с post данным из массива "date", результат ответа записался в $data["index"] далее сам метод инсталляции install_m

```
public function install_m ($data=null) {
file_put_contents($GLOBALS["foton_setting"]["path"]."/modul/modul/index.php",
$data["index"]);
//здесь мы можем выполнять любые действия с нашим массивом $data, в данном
случае мы записываем данные $data["index"] в файл
}
public function up(){
$connect=array("index"=>array("host"=>"(ваш
url)","method"=>"post","date"=>array(массив данных при инсталляции))); //
записываем данные ответа в переменную $data["index"]
return $connect;
}
```

после инициализации up был произведен запрос на наш url с post данным из массива "date", результат ответа записался в $data["index"], отличие от

метода install только в вызове, данный метод вызывается при обновлении модуля, а не при его инсталляции. Обновление доступно через модуль обновлений только при условии, что версия модуля на вашем сервере больше чем версия на сервере клиента.

```
public function up_m ($data=null) {
file_put_contents($GLOBALS["foton_setting"]["path"]."/modul/modul/index.php",
$data["index"]);
//здесь мы можем выполнять любые действия с нашим массивом $data, в данном
случае мы записываем данные $data["index"] в файл, тем самым обновляя его
}
public function del(){
$connect=array("index"=>array("host"=>"(ваш
url)","method"=>"post","date"=>array(массив данных при инсталляции))); //
записываем данные ответа в переменную $data["index"]
return $connect;
}
```

после инициализации del был произведен запрос на наш url с post данным из массива "date", результат ответа записался в $data["index"], отличие от метода install только в вызове, данный метод вызывается при удалении модуля, а не при его инсталляции. Удаление доступно только для установленных модулей, при удалении в корне модуля автоматически удаляется файл install._! Также удаление не означает полное удаление модуля, вы можете просто изменить данные, либо закрыть доступ, главное при условии, что вы оставляете ваш модуль соблюдать правила пустого шаблона модуля, для работоспособности всей системы.

```
public function del_m ($data=null) {
unlink($GLOBALS["foton_setting"]["path"]."/modul/modul/index.php");
$this->core->log('Файл /modul/modul/index.php удален');
//Вот так делать нельзя, ваш модуль будет неполным
//1. Вариант: вы удаляете весь модуль полностью
$this->core->dir_delete_foton($GLOBALS["foton_setting"]["path"]."/modul/modul");
$this->core->log('Модуль modul удален');
```

```
//и пользователю нужно будет скачать ваш модуль снова
//2. Вариант: вы очищаете файл index.php
file_put_contents($GLOBALS["foton_setting"]["path"]."/modul/modul/index.php","");
$this->core->log('Файл /modul/modul/index.php удален');
}
```

Структура пустого шаблона модуля для корректной работы системы

/modul/module

Структура папки

css – папка со стилями модуля, все файлы в корне этой директории подключаются автоматически при инициализации модуля

js - папка со скриптами модуля, все файлы в корне этой директории подключаются автоматически при инициализации модуля

item – директория содержит страницы модуля, например создав страницу /item/test.php в модуле htmlred мы можем обратится к ней в строке браузера так: /htmlred/test.modul

module_a.php — ajax методы модуля

module_c.php — контроллер модуля

module_m.php — модель модуля

index.php — главный файл модуля

module_a.php — содержит одноименный класс, по названию файла и наследуется от контроллера.

```
<?
class Module_a extends Module_c{

}?>
```

module_c.php — содержит одноименный класс, по названию файла и наследуется от модели.

```
<?
class Module_c extends Module_m{

}?>
```

module_m.php — содержит одноименный класс, по названию файла и наследуется от модели ядра.

```
<?
class Module_m extends Model{
public function name_m(){
            return 'Мой модуль'; //метод передает название модуля

}
public function modul_chmod(){
   return '1,2'; //метод передает список ролей пользователей, для которых данный модуль будет активным
}
public function version(){
return '1'; //версия вашего модуля целое или дробное число, например 1.5
}
 public function version_up(){
return array("host"=>"(ваш url)","date"=>array(массив данных при обновлении));
}

public function install(){
$connect=array("index"=>array("host"=>"(ваш url)","method"=>"post","date"=>array(массив данных при инсталляции))); //записываем данные ответа в переменную $data["index"]
);
return $connect;
}
public function install_m ($data=null) {
file_put_contents($GLOBALS["foton_setting"]["path"]."/modul/modul/index.php", $data["index"]);
//здесь мы можем выполнять любые действия с нашим массивом $data, в данном случае мы записываем данные $data["index"] в файл
}
public function up(){
```

```
$connect=array("index"=>array("host"=>"(ваш
url)","method"=>"post","date"=>array(массив данных при инсталляции))); //
записываем данные ответа в переменную $data["index"]
return $connect;
}
public function up_m ($data=null) {
file_put_contents($GLOBALS["foton_setting"]["path"]."/modul/modul/index.php",
$data["index"]);
//здесь мы можем выполнять любые действия с нашим массивом $data, в данном
случае мы записываем данные $data["index"] в файл, тем самым обновляя его
}
public function del(){
$connect=array("index"=>array("host"=>"(ваш
url)","method"=>"post","date"=>array(массив данных при инсталляции))); //
записываем данные ответа в переменную $data["index"]
return $connect;
}
public function del_m ($data=null) {

}
}?>
```

ЧПУ вашего модуля /ваш модуль.modul
ЧПУ для страниц вашего модуля в папке item /ваш модуль/страница в item.modul
Также директория модуля может содержать любое количество файлов и папок.

Работа с Ajax

Ajax методы к которым можно обращаться авторизованному в системе пользователю находятся в файле
/app/ajax/ajax_$GLOBALS["foton_setting"]['admindir'].php
Файл является контроллером, моделью этого файла является файл

/app/ajax/ajax_$GLOBALS["foton_setting"]['admindir']_m.php
ЧПУ для ajax /ваш метод.ajaxadmin
Ajax методы к которым можно обращаться не авторизованному в системе пользователю находятся в файле
/app/ajax/ajax_$GLOBALS["foton_setting"]['sitedir'].php
Файл является контроллером, моделью этого файла является файл
/app/ajax/ajax_$GLOBALS["foton_setting"]['sitedir']_m.php
ЧПУ для ajax /ваш метод.ajaxsite
Для кастомизации основного шаблона вы можете вносить правки в данные файлы, для правильной работоспособности шаблона не меняйте следующие методы файла /app/ajax/ajax_$GLOBALS["foton_setting"]['sitedir'].php:
autorizajax()
siteajax()
siteajax_vardump()
также для правильной работоспособности шаблона не меняйте методы файла /app/ajax/ajax_$GLOBALS["foton_setting"]['admindir'].php:
Для создания ajax запросов на сайте рекомендуется пользоваться методами
siteajax()
siteajax_vardump()
Для работы с данными методами необходимо создать метод контроллера любого из mvc шаблонов вашего сайта или интерфейса с названием начинающимся с
func_pub
например *func_pub_ajax(){*

```
if(isset($_POST['test'])){
            return $_POST['test'];
        }
}
```

если этот метод поместить в контроллер inform, то обращаться к методу через ajax jquery можно следующим способом

```
$.ajax({
    type: "POST",
    url: "/siteajax.ajaxsite",
    data: {controller:'inform',method:'func_pub_ajax',test:'Это тестовый шаблон'},
```

```
            success: function(data){
                alert(data);
            }
});
```

В данном случае test — post переменная, которая будет передана на метод контроллера, можно использовать любое количество таких post переменных.

Также вы можете создать директорию /app/view/{site}/ajax/

в которой создать директории class и view

Создайте файл test.php в каждой из этих директорий,

далее в файле /app/view/{site}/ajax/class/test.php напишите следующий код:

```
<?
class CAjax_test{
    public function __construct(){
        $this->core = new /Foton/Core;
    }
    public function mymethod(){
        return 'Hello World';
    }
}
```

В файле /app/view/{site}/ajax/view/test.php:

<?=$data['mymethod'];?>

Теперь при обращении по адресу /test.ajaxsite вы увидете строку «Hello World»

Создание виджетов

Виджеты хранятся в директории /dev/widget

Для создания виджета в директории $GLOBALS["foton_setting"]['sitedir']

Создайте директорию по названию вашего виджета.

В директории разместите файл:

index.php

Давайте создадим виджет с названием mywidget, тогда
/dev/widget/$GLOBALS["foton_setting"]['sitedir']/mywidget/index.php
Будет содержать

```
<? class Mywidget{
	function __construct(){
		$this->core =new /Foton/Core;;
	}
	public function index($str){
		return $str;
	}
}?>
```

Теперь данный виджет можно вызвать
$str='Hello World!';
$this->widget->mywidget->index($str);
Виджеты используются для отображения смешанного контента на сайте, как готовые блоки, например часы на js, или же калькулятор стоимости, виджеты рекомендуется использовать только если наличие смешанного контента не мешает вашей сео оптимизации и не затрудняет чтение кода. Также директория виджета может содержать любое количество файлов и папок.

Система интерфейсов

Система интерфейсов включает в себя несколько интерфейсов и стандартных mvc шаблонов.
Стандартная система интерфейсов включает в себя интерфейсы:
list
menu

taxonomy
И mvc шаблоны:
files
inform
users
workarea
zip
Таким образом система интерфейсов должна содержать хотя бы один интерфейс и хотя бы один mvc шаблон, например стартовую страницу workarea (или другое название в config.php)
При смене интерфейса файл Config.php стандартным модулем не меняется, поэтому необходимо соблюдение $GLOBALS["foton_setting"]['start_page'] в config.php и соответствие основному шаблону template.php.
Также интерфейсы в системе должны работать с хуком интерфейса interfaces();
Так как при отсутствии вашего хука интерфейса в шаблоне mvc сайта пользователя ваш интерфейс просто перестанет работать, а наличие метода interfaces(); обязательно для всех моделей mvc шаблонов, которые хранят данные через миграции.
При создании mvc шаблона, для подключения дополнительного стиля конкретно к вашему шаблону (странице) просто создайте файл в директории /view/$GLOBALS["foton_setting"]['admindir']/css/название вашего шаблона.css
и он автоматически подключится на вашей странице, то же относится и к шаблонам сайта, только путь будет уже /view/$GLOBALS["foton_setting"]['sitedir']/css/

Не забудьте, что названием шаблона является чистое название без префиксов _view,controller_,model_

Также можно подключить js скрипты, только директория уже будет /view/$GLOBALS["foton_setting"]['admindir']/js/, и аналогично /view/$GLOBALS["foton_setting"]['sitedir']/js

ЧПУ и оптимизация

Система сайта основана на mvc шаблонах, которые хранятся в директориях:
/app/model/$GLOBALS["foton_setting"]['sitedir']/
/app/controller/$GLOBALS["foton_setting"]['sitedir']/
/app/view/$GLOBALS["foton_setting"]['sitedir']/
/app/view/xml/
/app/view/json/
Роутинг системы сайта организован через таблицу router, для создания ЧПУ достаточно указать название представление view и ЧПУ routs
ЧПУ routs поддерживает следующие шаблоны:

%name%@a-z@ - name – переменная GET параметра, которой будет присвоено значение из ЧПУ, a-z — символы, которые должен включать параметр, если в параметре будут указаны другие символы, будет выведена ошибка 404
В данном случае ЧПУ должно обязательно включать параметр
Например с шаблоном news/%name%@a-z@/ адрес /news/hitech/ отработает код 200, и передаст $_GET['name']='hitech'; Просто /news// не передаст ничего и вернет ответ 200

%id%:0-9: - id – переменная GET параметра, которой будет присвоено значение из ЧПУ, 0-9 — символы, которые должен включать параметр, если в параметре будут указаны другие символы, будет выведена ошибка 404
В данном случае ЧПУ должно обязательно включать параметр
Например с шаблоном product-%id%:0-9:/ адрес /product-12/ отработает код 200, и передаст $_GET['id']='12'; Просто /product- не передаст ничего и вернет ответ 404
Вы можете создать любое количество ЧПУ для одного представления.
В приведенном примере, например url /news// не совсем корректен для сео, поэтому можно создать еще одно ЧПУ просто указав news/
При этом, если сам шаблон (название mvc шаблона) равно например mynews, то и по url /mynews/ страница тоже будет доступна. Для сео оптимизации это будль страницы, избежать дубля можно через контроллер, настроив редирект при соответственном url, а также указав в robots.txt

запрет индексации данной страницы, тогда по динамическому ЧПУ страница будет открыта для индексации, и даже помещена в карту сайта (подробнее о карте сайта читайте далее).

Создание xml файлов

Для создания xml версии страницы достаточно создать файл с названием вашего шаблона в директории xml, или скопировать из директории сайта /app/view/$GLOBALS["foton_setting"]['sitedir']/ в директорию xml, и отредактировать его под xml формат желательно с использованием шаблонизатора, для облегчения чтения разработчиком. Также вы можете воспользоваться модулем xml/json, только предварительно создайте мета-теги в разделе настройки системы->оптимизация для данного представления.

Далее, если у вас ЧПУ страницы product-%id%:0-9:/ или product-%id%:0-9:.html, то есть заканчивается на слеш или .html замените его на .xml и вам будет отображена страница из папки xml, при этом модуль и контроллер используется один и тот же.

Если вы используете прямое название вашего шаблона mvc, и в вашей системе ЧПУ нет дублей ЧПУ и представлений (когда ЧПУ одного шаблона равно названию другого шаблона mvc) вы можете указывать параметры через слеш: /test/param1/param2/ etc..., при этом вы будете оставаться в шаблоне test, и системой будут созданы переменные $_GET['2']='param1', $_GET['3']='param2' и т. д.

ЧПУ одинаково работает как для публичной так и для административной части.

Если у вас есть mvc шаблон для публичной части и такой же для административной, будут работать оба шаблона со своим ЧПУ, но если у вас ЧПУ одного шаблона равно названию другого шаблона в публичной или административной части, будьте готовы к неожиданностям.

Сео оптимизация:

Для управления основными мета-тегами страницы, такими как title,description,keywords используется таблица seo

Для работы в разделе настройки системы->оптимизация создайте запись с полями:

dir – директория сайта
view – название вашего mvc шаблона
title – заголовок страницы
keywords – ключевые слова страницы
description – описание страницы
В мета-тегах вы можете использовать шаблонизатор Foton для вывода динамического содержимого, в любом из мета-тегов доступен массив $data контроллера вашего представления.

Глобальный контроллер

Глобальный контроллер является объектом системы и подключается отдельно для административной и публичной части: находится соответственно в файлах /app/controller/admin_globals.php и /app/controller/site_globals.php, название класса одинаковое, поэтому одновременное подключение в обход системы неизбежно приведет к ошибке.
Обращение к методам этого контроллера происходит через $this->glob, например $this->glob->insert_logs();
Контроллер не обновляется системой и предназначен для написания собственных методов разработчиком. Объект $this->glob доступен во всей системе,где подключен класс ядра Model.

Кастомизация и события

С версии 0.7.0 появилась возможность кастомизировать полностью ваше приложение, для этого необходимо скопировать необходимые для кастомизации директории от корня сайта в созданную в корне сайта директорию, например «custom» и прописать в файле core/config.php:
$GLOBALS["foton_setting"]['custom'] = 'custom';
Также вы можете вынести эту директорию на один уровень с директорией вашего сайта, тогда нужно добавить в конце двоеточие:
$GLOBALS["foton_setting"]['custom'] = 'custom:';
Кастомизировать приложение также с версии 0.7.0 можно с помощью событий.

Создайте файл /dev/event.php со следующим содержимым:

```
<?
class Event{
      public function before_Core_abc09($x)
      {
        $x = base64_encode($x);
        return array($x);
      }
     public function after_Core_abc09($x)
      {
         return base64_decode($x);
      }
}
```

Теперь метод ядра abc09 не будет удалять все символы кроме латинских букв, цифр и нижнего подчеркивания так как мы перед предачей в метод строку кодировали, а перед выводом декодируем обратно для сохранения отображения.

Также в этом классе мы можем использовать методы работающие с obstart, для его включения нужно добавить в /core/config.php:

$GLOBALS["foton_setting"]['obstart']='Y';

Теперь вы можете заменять любые данные перед выводом в представлениях:

Для глобальной замены используйте Event->CallbackGlob:

```
public function CallbackGlob($text){
       return str_replace('<img src=','<img lazy="true" data-src=',$text);
 }
```

Для замены только для административной части используйте Event->CallbackAdmin:

```
public function CallbackAdmin($text){
       return str_replace('<img src=','<img lazy="true" data-src=',$text);
```

```
  }
```

Для замены конкретного шаблона используйте Event->Callback(MVC):
Например для главной страницы «html»:
public function Callbackhtml($text){
        return str_replace('<img src=','<img lazy="true" data-src=',$text);
  }
Также с версии 0.7.0 появилась возможность заменять методы ядра через файл /dev/custom_core.php
Укажите в нем код:

```
<?
trait Custom_core{

}
```

А также с помощью файла /dev/custom.php с кодом:

```
<? class Custom{

}
```

и создавайте свои методы, но ваши методы будут работать только, если вы не укажете явно $GLOBALS["foton_setting"]['version'] в core/config.php, в противном случае будет подключен трейт указанной версии ядра.

Проверка типов данных

С версии 0.5.2 появилась возможность проверять типы данных перед передачей в методы ядра Core или любого другого класса системы используя конструкцию вида:
$run = new /Foton/Run('_');
$this->core = $run->Core;
либо, например
$run = new /Fpton/Run(':');

$this->myclass = $run->MyClass;

В класс Run мы передаем разделитель, по которому он будет искать методы для нашего класса.

Для работы с типизацией разместите в файле /dev/type.php:

```
<?
class Type{
	public function Core_abc09()
	{
	  return ['array'];
	}
}
```

Используя метод, который должен принимать аргументом строку мы указываем на явное принятие массива, тогда данный код выполнен не будет и будет выведено предупреждение в консоли веб браузера:

abc09 arg: 0 ошибка типа - передан string требуется array

Обработчик Handler

Так как у framework Foton рендеринг и роутинг происходит в ядре, мы можем получать данные, которые поступаю сначала в контроллер, а затем в представление, и изменять их на всех этапах.

Для управления данными откройте файл /dev/handler.php

В момент формирования данных для контроллера вам доступны следующие свойства данного класса:

$this->dir – директория сайта

$this->format – формат представления (mvc,json,ajax,xml,tpl,face и т. д.)

$this->page – страница обработки

$this->get – массив get параметров

Для работы для какого-то определенного типа данных свойств необходимо создать метод с указанием названия свойства_ожидаемый результат. Например, для того, чтобы закрыть всю публичную часть сайта site достаточно создать метод

```
  public function dir_site(){
        $this->page = 'err503';
        $this->err503();
    }
//закрыть только для форматов xml
  public function format_xml(){
        $this->page = 'err503';
        $this->err503();
    }
//закрыть только для определенных страниц с форматом json
  public function format_json(){
        $arr_page_error = ['test1','test2'];
        if(in_array($this->page,$arr_page_error)){
                $this->page = 'err503';
                return $this->err503();
        }
        return $this;
    }
```

Методы выполняются обработчиком в следующем порядке

'dir','format','page','get'

На этапе отправки данных из контроллера в представление доступен следующий массив,

'dir','format','page','data'

$this->data – это массив $data, который передается из контроллера в представление.

Важно!!! Handler работает только для публичной части приложения, методы для свойств get и data не существуют, проверку этих данных можно выполнить в

одном из методов dir_,format_,page_

Обработчик Cron

Для создания очередей на обработку используйте метод ядра $this->core->cron($method,$data,$time)
$method – метод класса Cron расположенного в файле /dev/cron.php
$data – массив аргументов для данного метода
$time – периодичность запуска очереди для данного события, тип строка, «формат времени:time», например «i:5» означает, что данный метод будет выполняться раз в 5 минут, полный список обозначений форматов времени можно посмотреть здесь https://www.php.net/manual/ru/datetime.format.php
Свойство $this->count определяет количество запросов в пакете, то есть количество обработанных событий за 1 раз, по умолчанию 5.
Для запуска необходимо установить любую страницу сайта с get параметром foton-cron=$GLOBALS["foton_setting"]['multiplay'] на cron, либо создайте любое условие в файле /index.php на 28 строке и следуйте ему для вызова cron обработчика. Данный скрипт должен запускаться с периодичностью например раз в 1 минуту.

Вертикальный шардинг

Вертикальный шардинг позволяет обращаться к разным базам данных для хранения и модификации отдельных таблиц, базы данных могут находится на разных серверах.
Для того, чтобы на вашем проекте начал работу вертикальный шардинг необходимо создать базу данных на текущем, либо на другом сервере и перенести в нее стандартные таблицы, которые устанавливаются по умолчанию при установке системы, это необходимо для обработки типов данных в выводе интерфейсов системы.
После этого в файле /dev/sharding.php напишите следующий код:

```
<?
$arr =array(
    "site"=>
```

```
    array("html.mvc"=>array(
        'host' =>'ваш хост',
        "dbname"=>'ваша база',
        "login"=>'ваш логин',
        "pass"=>'ваш пароль'
        )
    ),
    "admin"=>array(
        "interface([^/]+)/html"=>array(
        'host' =>'ваш хост',
        "dbname"=>'ваша база',
        "login"=>'ваш логин',
        "pass"=>'ваш пароль'
        )
    )
);
```

site – массив для публичной части, admin — массив для административной части и содержит он в ключах шаблоны регулярных выражений.
Вместо html.mvc - мы пишем название представления и формат представления. Все элементы массива необязательные, здесь вы можете указать любые ключи суперглобального массива и на данной странице они будут переопределены, либо появятся.
Тоже самое касается административной части, только здесь вы указываете путь как в строке браузера от домена без слеша, и можете использовать регулярные выражения, так как обработка подключений к вашей таблице часто бывает специфичной. Теперь все таблицы в модели html будут созданы в новой базе и выводится в публичную часть будут именно из нее.

Работа с Memcached

Для работы с Memcached необходимо установить библиотеку Memcache и подключаете как в примере:

```
public function news(){
        $arr = array(
                'news',
                'sort'=>array('id'=>'DESC'),
                'where'=>array('=pub'=>'on')
            );
            return $this->core->getlist($arr);
 }
```

Например нам нужно закешировать результат метода ядра getlist, тогда меняем код метода вот так:

```
public function news(){
        $arr_cache = array('127.0.0.1','11211',50); //хост, порт, время кеширования в
секундах
        $obj = new /Foton/Cache($arr_cache);
        $arr = array(
                'news',
                'sort'=>array('id'=>'DESC'),
                'where'=>array('=pub'=>'on')
            );
            return $obj->Core->getlist($arr);
 }
```

Также вы можете использовать встроенное кеширование, для этого создайте объект new fCache(time) – где time – время кеширования в секундах, пример:

```
public function news(){
        $obj = new /Foton/fCache(50);
        $arr = array(
                'news',
                'sort'=>array('id'=>'DESC'),
                'where'=>array('=pub'=>'on')
            );
            return $obj->Core->getlist($arr);
```

```
}
```

Для очистки кеша добавьте в код $obj->clearCache(); - данный метод полностью очистит директорию кеша /system/cache/

```
public function clearCache(){
        $obj = new /Foton/fCache(50);
        $obj->clearCache();
}
```

Модульное тестирование

Модульные тестирование методов приложения выполняется с помощью класса Test, например нам нужно проверить работу метода ядра getlist, мы знаем, что результатом должен быть массив, при этом также мы знаем, что таблица заполнена хотя бы одной строчкой и соответственно должен быть элемент id>0, тогда проверяем тип результата:

```
// Здесь ключ любое название, значение — тип данных в php
        $type = [
                'output=>['data'=>'array'], //тип результата
                'input'=>['0'=>'array']//тип первого входного параметра
        ];
```

Элемент с id:

```
/*
```

Например, если результат метода массив и нам нужно проверить элемент $arr['test1']['test2'], то мы напишем ['test1:test2'=>'element']

Значение может принимать следующие флаги в начале:

= - проверка равно или нет

> - проверка меньше

< - проверка больше

! - не равно

% - содержит значение

*/

```
$value = [
    'output' =>['0:id'=>'=1'],//значение исходящего параметра
    'input'=>['0'=>'%test'], //значение входящих параметров, только для строк
    'value'=>[0=>'M:5:8'],//использует метод Core->rand() для создания рандомной строки**
    'range'=>[0=>[0,9]] //использует метод range для создания массива
];
```

** - флаги метода:

M – email

L – латинские символы

A – русские символы

0 — числа

Z – дополнительные символы

S – символы часто используемые в SQL-инъекциях

M:5 – означает выдать E-mail с длиной логина 5 символов

M:5:10 – означает выдать E-mail с длиной логина от 5 до 10 символов

M – выдает логин с длиной 10 символов

Создаем объект теста:

```
$test = new /Foton/Test($type,$value);
```

Теперь все методы можно начинать с него и далее ->класс_метода->метод():

```
$test->core->getlist(['catalogshop']);
```

А здесь выводим результат теста, тестов может быть несколько:

```
$test->test_log();
```

$test->test_return(); - вернет и очистит self::$test_log

$test->test_return(true); - только вернет self::$test_log

$test->clear_test() - очистить все свойства класса Test

Пример полностью:

```
$type = [
    'output'=>['data'=>'array'], //тип результата
    'input'=>['0'=>'array']//тип первого входного параметра
  ];

  $value = [
    'output' =>['0:id'=>'=1'],//значение исходящего параметра
    'input'=>['0'=>'%test'], //значение входящих параметров, только для строк
    'value'=>[0=>'M:5:8'],//использует метод Core->rand() для создания рандомной строки**
    'range'=>[0=>[0,9]] //использует метод range для создания массива
  ];
  $test = new /Foton/Test($type,$value);
  $test->core->getlist(['catalogshop']);
  $test->test_log();
  $test->clear_test();
```

Консольное приложение Foton

Обратится к консольному приложению Foton вы можете из командной строки перейдя в корень вашего сайта и набрав php foton, нажмите Enter и вам будет выведен весь список команд с пояснениями, это аналогично команде php Foton help:

up - обновление шаблонов

без параметров

opcahe - выполняет opcache preload файлов ядра

echo - вывод содержимого файла

model html site - выведет model с названием model_html.php из директории site

list - список файлов

model site - выведет список моделей из директории site

composer sdek:create – загрузит внутренний модуль sdek вместе с зависимыми модулями, sdek:update обновит его, sdek:drop удалит его

composer list – выведет список доступных модулей для загрузки

migrate - выполнить миграцию баз данных

interface directory - необязательные параметры

(interface: название метода модели после названия модели без _, например interface_sp)

(directory: название директории где искать модели, например site)

create site Test (-j) - создаст mvc шаблон Test в папке site -j - необязательная опция,при ее использовании создается модель работающая с миграциями

update site Test - обновит mvc шаблон Test в папке site

delete site Test - удалит mvc шаблон Test в папке site

widget - выведет список виджетов

module - выведет список внутренних модулей

type - выведет массив типов данных для работы с базой данных

diff Test - выведет изменения модели Test

print Test create - выведет текущее состояние модели Test со статусом create

m:del model table field1,field2 - без флага -t удаляем поля field1,field2 таблицы table в модели model

m:del model table custom_field -t - с флагом -t удаляем строку custom_field таблицы table в модели model

m:add model table field%f1,f2 -t - добавит строку field с полями f1,f2 в таблицу table модели model

m:add model table field%text%text - добавит поле field (название поля %формат вывода%формат хранения) в таблицу table модели model

m:up model table field%text%text - обновит поле field (название поля %формат вывода%формат хранения) в таблицу table модели model

m:drop model table - удалит таблицу table

m:create model table id,f1,f2%input,textarea,html%int,text,text%id - название полей через ,%форматы вывода через ,%формат хранения через , %поле ключа

m:alter Model table field1,field2:text - заменит поле field1 на field2 с типом text в таблице table модели model

Работа с транзакциями

При вызове методов модели типа «alter,del,interfaces» автоматически создаются транзакции в директории /app/model/transaction/, откатить изменения можно указав название модели транзакции с помощью консольного приложения Foton: php foton transaction имя транзакции
Также для тестирование запросов вы можете использовать метод ядра back($echo = 1, $stop = null) -
$echo – если 1 используется флаг FETCH_NUM
если 2 используется флаг FETCH_UNIQUE
если иное вызывается просто exec, по умолчанию $echo=1
$stop – по умолчанию null, если null выполняем откат rollback(), если другое значение сохраняем commit(), в случае ошибок выводится лог ошибок log.

Правила конвенции Foton

При разработке Framework Foton использовались принципы DRY, SOLID, KISS. В соответствии с этим при разработке своего приложения старайтесь следовать им:

- выделяйте функционал в отдельные объекты, либо методы.
- декомпозируйте решения (разбивайте сложные задачи на простые).
- пишите код настолько просто, чтобы и через год вы сами смогли в нем разобраться.
- при написании методов или объектов старайтесь предусмотреть возможность расширения вашего функционала без изменения вашего кода, либо с его минимальными изменениями.
- При указании путей к файлам работайте через $this->core->git() и указывайте пути от корня системы $GLOBALS["foton_setting"]['path']
- При использовании методов в своем методе используйте сквозные параметры, чтобы в идеале все параметры принимаемые внутренними методами вашего метода косвенно или напрямую зависели от переданной в метод сигнатуры.
- Также вы можете использовать CamelCase при написании своих методов или при обращении к методам ядра, для системы методы i_list и iList равнозначны.

Требования к синтаксису:

- Нежелательно использование альтернативного синтаксиса (через : ?,endif; и т. д.) для облегчения чтения кода и определения пределов алгоритмов в различных IDE, по стандартам конвенции Foton используется стандартный синтаксис if(){}else{],foreach(){},for(){}.
- Не используйте смешанный код (в одном файле css,js,php,html код, это затрудняет поддержку и чтение вашего кода)
- Соблюдайте чистоту кода и отступы.
- При создании представления внешней части вашего приложения используйте шаблонизатор Foton, либо другие шаблонизаторы.
- По возможности не используйте операторы switch case, особенно в сложных алгоритмах, это усложняет чтение кода.

Другие требования:

- Файлы должны быть в кодировке UTF-8 без BOM-байта.
- Имена класса и его метода не должны совпадать.
- Не используйте слишком часто статические методы, когда можно обойтись без них, это увеличивает время работы ваших методов, системе каждый раз приходится инициализировать ваш класс.
- При использовании большого объема данных для расчетов используйте сохранение данных в переменные, используйте обратный отчет для перебора данных.
- Используйте буферизацию ob_start() только через стандартные средства системы, иначе это может усложнить рефракторинг вашего приложения в дальнейшем.
- Не используйте метки и прерывания php без веской на то причины. Ваш код может быть переписан и ваши метки приобретут совсем другое значение.
- При создании сигнатур методов присвойте им default значения, и проверяйте значения сигнатур на входе, это поможет избежать ошибок при копировании метода разработчиком в контроллер, либо при неправильном использовании.
- Используйте обработку ввода данных в модели, либо с помощью класса Validate и с использованием соответствующих методов ядра при получении $_REQUEST данных, и других незащищенных данных

пользователя перед обработкой контролером.

- При изменении метода в дочернем классе используйте по возможности конструкцию по типу такой:

```
public function method() {
    $this_method = parent::method();
    $this_method = $this->new_method($this_method);
    return $this_method;
}
```

- То есть, вместо копирования кода или полного переопределения мы получили метод родителя и обработали его в соответствии с новыми требованиями, таким образом вы поддержите код предыдущего разработчика и следующему будет проще понять логику изменения вашей архитектуры.
- Если вы не хотите видоизменения вашего метода финализируйте его (указав final) и укажите эти методы в комментариях к вашему классу.
- При написании внутренних модулей нужно пользоваться стандартными методами ядра для подключения объекта класса $this->core->m_obj()
- Нельзя переопределять методы ядра, или переписывать их, так как при обновлении изменения будут удалены и следующему разработчику это может составить большие трудности, когда привычный метод начнет проявлять себя не так как должен, для изменения вы можете воспользоваться trait Core_Custom, либо для переопределения независимых методов классом Custom.
- Общие методы для всей системы вы можете создать в глобальном контроллере, при обращении к вашему методу разработчику нужно будет использовать конструкцию $this->glob->вашметод().

Зарезервированные элементы системы

1. Объекты системы
    - $this->get_post

- $this->db
- $this->obj_m
- $this->mod
- $this->widget
- $this->request
- $this->glob
- $this->core
- $this->git()
- $this->post_replace()
- $this->post_get()
- $this->select_db()

2. Переменные сессии
   - $_SESSION['login']
   - $_SESSION['gittest']
   - $_SESSION['login2']
   - $_SESSION['chmod_id']
   - $_SESSION['speed_test']
   - $_SESSION['multiplay']
3. Глобальные переменные

   $GLOBALS["foton_setting"]["foton_setting"] – содержит следующие элементы:
   - ["host"] — хост подключение к базе данных
   - ['obstart'] = "Y/N" - включение выключение обработки ob_start() представлений, имеет хуки для изменения выдаваемого контента в файле dev/event.php - Event->CallbackGlob, Event->CallbackAdmin, Event->Callback(MVC) (MVC) – название MVC, например html.
   - ["dbname"] — имя базы данных
   - ["login"] — логин пользователя базы данных
   - ["pass"] — пароль пользователя базы данных
   - ["sql"] – сервер баз данных, может принимать значения mysql,pgsql,lite
   - ["license"] — номер лицензии ядра
   - ["coref"] — версия ядра
   - ['version']='Core82'; - при указании версии дополнительно подключается соответствующая версия ядра, если на момент изменения она trait версии содержится в ядре /core/core.php, обычно

указаны в начале файла.

- ['key'] — директория архивов сайта
- ["path"] — корневой путь к сайту
- ["sitedir"] — директория сайта
- ["admindir"] - директория админ. Панели
- ['admin'] — путь для входа в административную панель
- ['time'] - начало времени работы скрипта
- ['sizelog'] - максимальный размер сохраняемых файлов логов в день
- ["main"]='html'; - mvc шаблон главной страницы публичной части приложения.
- ["main"] - главная страница сайта
- ['http'] — протокол
- ['value_log'] - массив имен (name) POST данных сохраняемых в логах
- ['templates'] — путь к основному шаблону системы
- ['multiplay'] - множитель сессии для защиты от кражи сессии
- ['interface'] - основной интерфейс
- ['size_file'] - максимальный размер в меабайтах для загрузки файлов
- ['format'] — массив разрешенных форматов файлов для загрузки
- ['max_list'] - максимальное число записей для вывода без интерфейса
- ['start_page'] - главная страница при входе в админ. Панель
- ["error404"] — страница 404 ошибки
- ["git"] - включаем git Y/N
- ["lang"] — язык системы
- ["lib"]='/core/lib/lib'; - путь к библиотеки обработки системы, можно изменить на свой скопировав соответствующий файл для вашей кастомизации.
- ["adapter"]='/core/lib/adapter'; - путь к адаптеру обработки системы, можно изменить на свой скопировав соответствующий файл для вашей кастомизации.
- ["preload"] = "Y/N" - если включен идет обработка ядра через core/lib/run.php – поддерживаются события в файле dev/type.php,/dev/event.php,/dev/custom.php, работает только с версии php 7.0.1
- ['ifile'] = $GLOBALS["foton_setting"]['path'].'/app/controller/'. $GLOBALS["foton_setting"]['admindir'].'/file'; - необходим для работы с

файловым интерфейсом, вы можете указать здесь свой путь, предварительно создав соответствующую директорию.

- ['custom'] – необязательный параметр системы, если он указан, то файлы вашего приложения будут браться из указанной директории, например если вы укажете $GLOBALS["foton_setting"]['custom'] = 'mydir'; , создайте директорию mydir в корне вашего сайта и скопируете туда ваше приложение, если все файловые пути вашего приложения работают через git(), то оно будет работать именно из этой директории, в противном случае могут возникнуть ошибки. Если вы хотите вынести вашу кастомную директорию на один уровень с директорией сайта просто укажите в конце :, вот так $GLOBALS["foton_setting"]['custom'] = 'mydir:'; Не забывайте о правах на созданную директорию, она должна быть под тем же пользователем, что и директория вашего сайта.
- ['winrar'] – необязательный параметр, указывает путь к исполняемому файлу winrar для создания бекапов системы в ОС Windows, если он не указан используется путь по умолчанию - C:\Program Files\WinRAR\rar.exe
- ['smtp'] = ['host'=>'ssl://smtp.yandex.ru','port'=>465,'login'=>'login@yandex.ru','pass'=>'password']; - необязательный параметр, если указан mail_foton() будет отправлять E-mail через SMTP.
- ['water_mark'] – необязательный параметр, указывает путь к водяному знаку.
- ['image_def'] — необязательный параметр, указывает путь к фото, если фото не найдено.
- ['get_unset'] – если равна 'Y', в публичной части приложения будут не доступные стандартные get параметры через ? - для безопасности лучше поставить эту переменную активной, тогда все get параметры будут формироваться только из безопасных ЧПУ в системе.
- ['debag'] – при установке этого ключа в массиве с любым значением будет включена автоматическая обработка кода и вывод вызванных файлов и методов контроллера и ядра, а также время вызова в виде unix метки времени.
- Классы — Core,Main,Parents,Custom,Type,Run,Event,Test,View,Model,

Cache,Config,Router,Render,RenderLib,dataReturn,dataReturnLib,Mod, Widget,Validate.

4. Зарезервированное ЧПУ
    - /core/*
    - /app/*
    - /system/*
    - /dev/*
    - /git.php
5. Зарезервированные форматы файлов ЧПУ
    - .ajax
    - .ajaxsite
    - .ajaxadmin
    - .modul
    - .tpl
    - .face
    - .xml (возможно использовать путь до физического файла)
    - .json (возможно использовать путь до физического файла)

Особенности системы

Система содержит хук контроллера dir(), если данный метод в контроллере отсутствует система будет искать представление в той же папке, что и контроллер. Если контроллер находился по абсолютному пути /app/controller/admin/, то и представление система будет искать в директории /app/view/admin/.Если же данный метод задан, например:

```
public function dir(){
        return 'mydir';
}
```

то система будет искать представление для данного контроллера в

директории /app/view/mydir/

Ядро системы имеет 3 финализированых метода, заданных как public funal function:

m_method

m_class

m_obj

Эти методы нельзя переопределять, в противном случае произойдет ошибка.

Все эти 3 метода последним необязательным аргументом принимают массив или переменную как сигнатуру класса, при этом конструктор класса должен иметь не более одного обязательного аргумента, передать несколько аргументов в виде массива не получится, так как класс вызывается явно **new Class($argument);**

Для установки нового интерфейса в систему достаточно скачать интерфейс, разархивировать в директорию /system/api/tpladmin/, выбрать данный интерфейс в модуле «Интерфейсы» и установить его.

Для установки нового шаблона сайта в систему достаточно скачать шаблон, разархивировать в директорию /system/api/tplsite/, выбрать данный шаблон в модуле «Шаблоны сайта» и установить его.

Для установки еще одного интерфейса кроме существующего, например для другого пользователя достаточно зайти в модуль интерфейсы, далее открыть и отредактировать файл /core/config.php

Заменить ["admindir"]='admin'; на ["admindir"]='newdir';

newdir – ваша новая директория

После этого вернутся в модуль интерфейсы, и выбрав нужный интерфейс просто установить его.

После этого установите директорию по умолчанию как вам нужно, например:

```
If($_SESSION['login']=='admin'){
    $GLOBALS["foton_setting"]["admindir"]='admin';
}
else{
    $GLOBALS["foton_setting"]["admindir"]='newdir';
}
```

Для установки еще одного шаблона сайта кроме существующего, например для другого домена достаточно зайти в модуль шаблоны сайта, далее открыть и отредактировать файл /core/config.php

Заменить $GLOBALS["foton_setting"]["sitedir"]='site'; на

$GLOBALS["foton_setting"]["sitedir"]='newdir';

$GLOBALS["foton_setting"]["dbname"]='newdb';

$GLOBALS["foton_setting"]["login"]='newlogin';

$GLOBALS["foton_setting"]["pass"]='newpass';

предварительно создав эту базу с пользователем и скопировав туда таблицы role и user_inc

newdir – ваша новая директория

После этого вернутся в модуль шаблоны сайта, и выбрав нужный шаблон просто установить его.

После этого установите директорию по умолчанию как вам нужно, например:

```
If($_SERVER['SERVER_NAME']=='newdomen.ru'){
    $GLOBALS["foton_setting"]["sitedir"]=newdir';
    $GLOBALS["foton_setting"]["dbname"]='newdb';
    $GLOBALS["foton_setting"]["login"]='newlogin';
    $GLOBALS["foton_setting"]["pass"]='newpass';
}
else{
    $GLOBALS["foton_setting"]["sitedir"]='site';
    $GLOBALS["foton_setting"]["dbname"]='db';
    $GLOBALS["foton_setting"]["login"]='login';
    $GLOBALS["foton_setting"]["pass"]='pass';
}
```

При добавлении шаблона, также можно просто скопировать файл из другого шаблона dump.sql в директорию того, который хотите установить, или же сделать дамп своей системы и сохранить его в директории шаблона с названием dump.sql.

Это можно делать, только если структура баз идентична.

Система автоматически ищет регионы сайта (шапка и футер) head.php и foot.php в директории /app/view/$GLOBALS["foton_setting"]["sitedir"], и если не находит выполняет поиск в корневой директории /app/view/, при этом файлы подключаются на уровне контента $data->html(); в основном шаблоне /app/view/template.php, для отключения регионов на странице воспользуйтесь методом region в контроллере своего шаблона mvc:

```
<?
public function region(){
	return false;
}
?>
```

Тогда в представлении вашего контроллера регионы подключены не будут.
Если вы хотите выводить одинаковые шапку и футер для сайта и интерфейса удалите файлы
head.php и foot.php из их директорий, например из /app/view/site и /app/view/admin/ и создайте файлы head.php и foot.php в корне директории /app/view/.
Также если вы хотите переопределить шаблон template.php для публичной части приложения, создайте в директории /app/view/site/ файл template.php и он будет подключен автоматически.
Для удаления шаблона сайта удалите следующие директории и файлы:
/app/ajax/шаблон вашего сайта/
/app/view/шаблон вашего сайта/
/app/lang/шаблон вашего сайта/
/app/controller/шаблон вашего сайта/
/app/model/шаблон вашего сайта/
/app/controller/шаблон вашего сайта_globals.php
/dev/widget/шаблон вашего сайта/
/system/api/tpl/шаблон вашего сайта/
/system/api/php/шаблон вашего сайта/

а также базу данных данного шаблона, либо просто замените шаблон через модуль шаблоны сайта, старая информация автоматически удалится.
Для удаления интерфейса удалите следующие директории и файлы:
/app/ajax/ваш интерфейс/
/app/view/ваш интерфейс/
/app/lang/ваш интерфейс/
/app/controller/ваш интерфейс/
/app/model/ваш интерфейс/
/app/controller/ваш интерфейс_globals.php
/system/api/tpl/ваш интерфейс/
а также базу данных данного шаблона, либо просто замените шаблон через модуль шаблоны сайта, старая информация автоматически удалится.
Для подключения отдельного шаблона к странице укажите в файле /app/view/tpl-ini.php ЧПУ и адрес шаблона для представления, которое должно отображаться в данном шаблоне.
Например для отображения представления workarea в новом шаблоне /app/view/work.php, создайте данный шаблон в котором работают объекты $data — методы роутера и рендера, $m — методы модели, $c – методы ядра.
И пропишите в /app/view/tpl-ini.php
/workarea/=app/view/work.php
Для того, чтобы все страницы, содержащие в ЧПУ /interface_list/ использовали ваш шаблон напишите /interface_list/(.*)=app/view/work.php

Все основные настройки системы находятся в модели /app/model/вашсайт/model_html.php, даже если вы изменили главную страницу вашего сайта в настройках не удаляйте данную модель и не меняйте ее содержимое, если не уверены в том, какие последствия для системы это может повлечь и если в этом нет необходимости.