



IPLOGIC IPLF FRAMEWORK V1.X

Руководство разработчика

Создание проектов на основе FrameWork IPLF

В данном руководстве представлены описания основных принципов разработки web приложений и описание классов и методов IPLF

Rev 3
27.04.2015

Оглавление

| | |
|---|----|
| Описание iPloGic IPLF FrameWork 1.x | 3 |
| Основы разработки приложений | 4 |
| Этапы разработки | 4 |
| Термины и определения | 4 |
| Виды архитектуры | 5 |
| Файлы и директории | 5 |
| Области доступа | 6 |
| Индексный файл | 9 |
| Создание компонента | 13 |
| Файл модели | 13 |
| Файл инициации | 13 |
| Установка компонента (раздела) | 14 |
| Написание служебных скриптов | 15 |
| Описание файлов настройки | 16 |
| Файл settings.php | 16 |
| Файл config.php | 16 |
| Базовые классы (файл base.classes.php) | 19 |
| Класс FRAME_CORE | 19 |
| Класс CONTROLLER | 21 |
| Работа с шаблонами (файл template.classes.php) | 23 |
| Переменные и конструкции | 23 |
| Класс VIEW | 24 |
| Пример | 26 |
| Инструменты работы с базами данных (файл db.classes.php) | 29 |
| Класс DB | 29 |
| Пример | 31 |
| Инструменты работы с пользователями (файл user.classes.php) | 33 |
| Класс USER_STATIC | 33 |
| Класс USER | 34 |
| Примеры | 36 |
| Инструменты работы с формами (файл form.classes.php) | 38 |
| Класс FORM | 38 |
| Типы полей и атрибуты | 41 |
| Шаблоны форм | 42 |

| | |
|---|----|
| Пример | 43 |
| Инструменты работы с изображениями (файл images.classes.php) | 46 |
| Класс IMAGE | 46 |
| Примеры | 52 |
| Создание постраничного вывода (файл pagination.classes.php) | 55 |
| Класс PAGINATION | 55 |
| Шаблоны по умолчанию и переменные | 57 |
| Примеры | 59 |
| Инструменты работы с древовидными структурами (файл tree.classes.php) | 61 |
| Класс TREENODE | 61 |
| Класс TREE | 62 |
| Примеры | 64 |
| Сервисные классы (файл services.classes.php) | 67 |
| Класс TIMER | 67 |
| Класс LOGGER | 67 |
| Пример | 68 |
| Статические функции (файл functions.classes.php) | 69 |
| Класс FUNC | 69 |
| Стандартные SQL таблицы | 74 |
| Полный список констант и базовых параметров | 78 |
| Задаваемые константы | 78 |
| Формируемые константы | 79 |
| Базовые параметры | 79 |
| Журнал изменений | 81 |

Описание iPloGic IPLF FrameWork 1.x

iPloGic IPLF FrameWork (далее IPLF) – мощный инструмент для создания веб-приложений любой сложности и нагруженности. Он представляет собой набор взаимосвязанных классов PHP для быстрой и удобной разработки и интеграции. В его состав входят как инструменты непосредственного построения структуры приложения по заданной архитектуре, так и дополнительные библиотеки для упрощения и стандартизации разработки компонентов.

Архитектура приложений, разработанных на основе IPLF построена по схеме MVC. При этом возможно создание простых одномоделных разделов или сложных разветвленных компонентов.

Обработка получаемого адреса и гибкие настройки позволяют усилить защиту приложения от взлома.

Возможности:

- ✓ Определение профиля текущего режима приложения
- ✓ Формирование html кода из шаблонов
- ✓ Работа с SQL базами данных
- ✓ Авторизация и аутентификация пользователей
- ✓ Разграничение областей доступа пользователей
- ✓ Построение и обработка форм
- ✓ Построение страничных списков
- ✓ Обработка и создание изображений
- ✓ Обработка и создание деревьев элементов
- ✓ Создание логов
- ✓ Расчет времени выполнения участков скриптов
- ✓ Использование SEF ссылок
- ✓ Наличие библиотеки, расширяющей стандартный PHP функционал

Основы разработки приложений

Этапы разработки

Разработка приложений на основе IPLF состоит из следующих этапов:

- ✓ Установка FrameWork (копирование файлов IPLF)
- ✓ Определение файловой структуры и архитектуры приложения
- ✓ Правка параметров в файлах настройки
- ✓ Создание базы данных и необходимых таблиц (если предполагается ее использование)
- ✓ Создание индексного файла
- ✓ Создание компонентов и разделов приложения

Термины и определения

Для лучшего понимания опишем те термины, которые вы будем в дальнейшем использовать для описания процедур и классов, так как в различных системах встречается разное использование одних и тех же определений.

Приложение

Общее определение разрабатываемого ресурса, построенного на одном экземпляре FrameWork, включая все его функциональные единицы и вспомогательные скрипты.

Компонент (component)

Логически и функционально законченная часть сайта, выполняющая определенные задачи. При PFC архитектуре содержит только один файл модели находящийся в корне директории компонентов. Для PFCS это директория содержащая файл инициации разделов и модели разделов. Так компонент, например, может содержать все скрипты блога или интернет-магазина.

Раздел (section)

Представляет собой файл модели текущего представления компонента. Содержит в себе класс, вызываемый ядром IPLF при обращении к компоненту.

Шаблон (template)

Файл содержащий html представление (view в схеме MVC) и используемый для формирования результирующего html кода контента компонента.

Лэйаут (layout)

Шаблон, используемый для построения результирующего html кода страницы типовой для сайта или его части.

Виды архитектуры

Общая архитектура IPLF построена на основе схемы MVC, но при этом имеет два вида ее реализации. Для простоты в дальнейшем будем называть их видами архитектуры IPLF.

Существует два вида архитектуры: PFCS и PFS. Разница между ними заключается в способе вызова модели контроллером.

Модель раздела (идентификатор) определяется контроллером исходя из адреса страницы. Это первый после корневого URL раздел ссылки. Например, для сайта, расположенного в корневой директории, при ссылке «<http://site.com/page/some/>» идентификатором раздела будет «page». При использовании псевдонимов страниц идентификатор извлекается из обработанной ссылки.

PFCS

При этом виде архитектуры обращение контроллера к модели происходит через файл инициации (init.php), в котором находятся массивы соответствия идентификаторов разделов и файлов их моделей.

Компонент в этом случае представляет собой директорию, имеющую название идентификатора компонента и расположенную в директории нахождения компонентов. Она содержит файл init.php, файлы моделей и любые другие файлы необходимые для работы компонента.

Поиск необходимого компонента ведется при помощи файла кеширования схемы соответствия идентификаторов разделов и компонентов components.cch, находящегося в корне директории нахождения компонентов. Это значительно ускоряет работу контроллера.

Эта архитектура отличается структурированностью скриптов и подходит для создания при создании приложений любой сложности.

PFS

Упрощенная архитектура, предназначенная для разработки простых приложений, которые не предполагают использование большого количества различных разделов.

При данной архитектуре компонентом является файл модели раздела, находящийся в корне директории нахождения компонентов. Этот файл должен иметь название, совпадающее с идентификатором раздела. Например, для идентификатора раздела «page», файл будет иметь название page.php.

Файлы и директории

Рассмотрим минимально необходимые для работы IPLF директории и файлы, которые должны быть созданы на сервере для дальнейшей разработки приложения.

```
/
  components
  iplf
  view
    templates
    layouts
  404.php
  index.php
  settings.php
  sqlerror.php
```

components - директория размещения компонентов. Название может быть изменено на любое другое, при условии изменения соответствующего параметра в файле config.php.

iplf (или IPLF) - директория скриптов IPLF (не указаны в списке). Название может быть изменено без дополнительных действий.

view – общая директория нахождения файлов создания внешнего вида страниц (шаблоны, лейауты и т.п.). Название может быть изменено на любое другое, при условии изменения соответствующего параметра в файле config.php.

templates – директория шаблонов компонентов области public. Находится корне директории view. Название фиксировано.

layouts – директория лейаутов области public. Находится корне директории view. Название фиксировано.

404.php - файл, включаемый при отсутствии раздела или отдельного адреса. При его отсутствии выводится системное сообщение.

index.php – основной индексный исполняемый файл.

settings.php – файл настроек приложения. Обязательный файл.

sqlerror.ru – файл, подключаемый при ошибке подключения к базе данных SQL. При его отсутствии выводится системное сообщение.

Области доступа

В IPLF существует понятие «Область доступа» (AccessArea). Область доступа представляет собой раздел сайта, обособленный от основного контента. Например, это может быть панель администратора или личный кабинет пользователей. Эти области имеют собственные шаблоны и ограничены в доступе.

По умолчанию на сайте существует одна область **public**. Остальные области создаются отдельно и должны быть расположены в собственных разделах (директориях). Но при этом формирующие страницы этой области разделы находятся в общей директории компонентов, а в выделенной директории находятся темы оформления, файл авторизации и файлы, используемые только для этой области.

Лучше всего показать создание области доступа на примере. Предположим, что нам необходимо добавить административный раздел на сайт, работающий по архитектуре PFS. В этой области будет две страницы – главная (файл раздела `home.php`) и настройки (файл `settings.php`).

Изначально имеем следующую структуру директорий сайта (чтобы не путаться возьмем минимальную конфигурацию):

```
/
  components      # директория компонентов
  iplf             # директория скриптов IPLF
  view            # директория шаблонов области public
    templates     # директория шаблонов компонентов области public
    layouts       # директория лейаутов области public
```

Создаем директорию `administrator`, которая будет являться директорией области `admin`. В созданную директорию добавляем поддиректорию `view` и вложенные в нее `templates` и `layouts`

Если сейчас попробовать зайти по адресу `site.com/admin/` или `site.com/administrator/`, то эти адреса будут обработаны как попытка зайти в раздел `admin` или `administrator` области `public`. Для того, чтобы появилась необходимая область доступа надо ее определить в соответствующих массивах в файле `/iplf/config.php`.

\$wfconfig_access_areas - массив определяющий особые области доступа и разделы, в которых они расположены. Область `public` здесь не определяется. Область доступа является константой. В нашем случае он будет выглядеть так:

```
$wfconfig_access_areas = Array(
    'administrator' => 'admin'
);
```

\$wfconfig_access_areas_ids – массив, определяющий идентификаторы областей доступа, включая область `public`. Он примет вид:

```
$wfconfig_access_areas_ids = Array(
    0 => 'public',
    1 => 'admin'
);
```

Теперь ссылка `site.com/admin/` ведет нас в новую область (но при этом `site.com/administrator/` все также будет рассматриваться как ссылка области `public`).

Далее необходимо создать файл авторизации для данной области.

area_authorization.php – файл, находящийся в корневой директории (для нашего примера `administrator`) любой области доступа кроме `public` и вызываемый в случае отсутствия авторизации пользователя или отсутствия у него необходимых прав. Если он отсутствует, то будет выводиться системное сообщение о запрете доступа в раздел.

Доступ пользователей к областям и разделам определяется в стандартных таблицах USERS_GROUPS и USERS_GROUPS_ACCESS.

Теперь осталось только добавить файлы home.php и файл settings.php и шаблоны с лэйаутами для них. Получаем следующую структуру директорий:

```
/
  administrator      # директория области доступа admin
    view             # директория шаблонов области admin
      templates      # директория шаблонов компонентов области admin
      layouts        # директория лэйаутов области public
  components         # директория компонентов (включая home.php и
                      # settings.php)
  iplf               # директория скриптов IPLF
  view               # директория шаблонов области public
    templates        # директория шаблонов компонентов области public
    layouts          # директория лэйаутов области public
```

Все. Раздел администратора создан. Теперь перейдя по ссылке site.com/admin/settings/ и зарегистрировавшись с соответствующими правами, мы попадем на страницу настроек административного раздела.

Индексный файл

Индексный файл – основной исполняемый файл, вызываемый при обращении к приложению. Он создается разработчиком и выполняет следующие функции:

- ✓ Подключение файлов настройки
- ✓ Предварительная проверка входных параметров и их обработка (например, перенаправление или выбор конфигурации)
- ✓ Подключение обязательных файлов IPLF
- ✓ Подключение библиотек IPLF, используемых в большинстве компонентов
- ✓ Инициация ядра и базы данных
- ✓ Установка параметров работы
- ✓ Получение результата формирования htmlкода страницы и ее вывод
- ✓ Дополнительные служебные действия

Пример файла (авторизация без использования cookie):

```
<?

// начинаем сессию
session_start();

// подключаем файлы настройки
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'errors_mode.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'settings.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'config.php');

// выявление особых условий и их реализация до начала построения страницы
//(редиректы, сообщения и т.п.)
if ( $GET['redirect']=='yes' ) { header("Location:
http://site.com/redirect.php "); }

// устанавливаем кодировку документа
header( 'Content-Type: text/html; charset='.DOCUMENT_CHARSET );

// подключение основных файлов
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'base.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'db.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'user.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'functions.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'template.classes.php');

// подключаем дополнительные библиотеки
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'services.classes.php');
```

```

// устанавливаем таймер выполнения скрипта (служебное действие)
$timer = newTIMER();
$timer->SetStart();

// подключаемся к ядруIPLF
$score = FRAME_CORE::getInstance();

// иницилируем базу данных
$score->DBInit();

// отменяем логирование ошибок SQL
FRAME_CORE::DB()->log_errors = false;

// запуск построения профиля ядра
$score->Init();

// формирование и получение html кода страницы
$html_result = $score->Execute();

// остановка таймера выполнения
$timer->SetStop();

// внесение времени выполнения на страницу (определение имени переменной
// произвольное, в данном случае '##total_time##')
$html_result=str_replace('##total_time##',$timer->GetResult(),$html_result);

// окончательный вывод html
echo $html_result;

// окончание работы
die();

?>

```

Следует обратить внимание, что подключение к ядру происходит именно через метод FRAME_CORE::getInstance(). Это обязательное условие, обеспечивающее нормальную работу всех классов IPLF.

Пример файла (авторизация с использования cookie):

```

<?

// подключаем файлы настройки
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'errors_mode.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'settings.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.DIRECTORY_SEPARATOR
.
'config.php');

```

```

// выявление особых условий и их реализация до начала построения страницы
// (редиректы, сообщения и т.п.)
if ( $_GET['redirect']=='yes' ) { header("Location:
http://site.com/redirect.php "); }

// подключение основных файлов
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'base.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'db.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'user.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'functions.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'template.classes.php');

// подключаем дополнительные библиотеки
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'services.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'pagination.classes.php');
require_once(dirname(__FILE__).DIRECTORY_SEPARATOR.'IPLF'.
DIRECTORY_SEPARATOR.'form.classes.php');

// подключаемся к ядруIPLF
$core = FRAME_CORE::getInstance();

// инициализируем базу данных
$core->DBInit();

// отменяем логирование ошибок SQL
FRAME_CORE::DB()->log_errors = false;

// проверка наличия сессии и ее запуск при необходимости
if ( isset($_COOKIE['session']) )
{ USER_STATIC::StartSession($_COOKIE['session']); }

// устанавливаем кодировку документа
header( 'Content-Type: text/html; charset='.DOCUMENT_CHARSET );

// запуск построения профиля ядра
$core->Init();

// формирование и получение html кода страницы
$html_result = $core->Execute();

// окончательный вывод html
echo $html_result;

// окончание работы
die();

?>

```

При использовании метода авторизации через cookie подключение сессии в начале файла не производится. Вместо этого после инициации базы данных вставляется строка проверки и подключения сессии через cookie. Только после этого отправляются заголовки в браузер и запускается метод FRAME_CORE::Init() ядра.

Создание компонента

Компонент – базовая единица в работе приложения. Все компоненты расположены в директории «components» (название можно изменить, отразив это изменение в файле config.php). Компонент может представлять собой файл модели раздела (архитектура PFS) или директорию, содержащую модели разделов компонента (архитектура PFCS).

Файл модели

Файл модели раздела представляет собой основной исполняемый PHP файл, который иницирует работу всех необходимых для построения страницы раздела по текущим параметрам.

Этот файл должен содержать класс **Component** дочерний для класса CONTROLLER. Метод Component ::__construct должен формировать четыре переменные:

- ✓ \$this->content – html текст раздела, который в дальнейшем будет вставлен в лэйаут
- ✓ \$this->title – метатег title
- ✓ \$this->description – метатег description
- ✓ \$this->keywords – метатег keywords

Пример простейшего файла модели:

```
<?
class Component extends CONTROLLER
{
    function __construct(){
        $this->content = 'Hello world!';
        $this->title = 'New page';
        $this->description = 'New page of this site';
        $this->keywords = 'site, hello';
    }
}
?>
```

Файл модели может содержать любые другие методы и классы кроме обязательных.

Файл инициации

Файл инициации init.php используется при архитектуре PFCS. Он располагается в директории компонента и содержит массив(ы) определяющие соответствие идентификаторов разделов и имен файлов моделей, являющихся их обработчиками. Обработчики должны находиться в той же директории.

Массив соответствия имеет имя «models» и является двухмерным. Первым параметром является область доступа, вторым идентификатор раздела.

Пример:

```
<?
$models['public'] = Array(
    'home'=>'home.php',
    'page'=>'page.php'
);

$models['admin'] = Array(
    'home'=>'_home.php',
    'settings'=>'_settings.php'
);

?>
```

В приведенном примере компонент используется в областях «public» и «admin» и содержит четыре файла моделей.

Установка компонента (раздела)

Для установки компонента (раздела) необходимо произвести следующие действия:

1. Поместить файл или папку компонента в директорию «components»
2. При использовании стандартных таблиц компонентов внесите информацию о компоненте и разделе в соответствующие таблицы
3. Удалите файл components.ssh из директории «components»

При создании и установке новых компонентов будьте внимательны. Идентификаторы компонентов и разделов должны быть уникальными.

Написание служебных скриптов

В данном случае под понятием скрипта мы будем понимать файл (группа файлов), который не входит непосредственно в структуру приложения и сам не участвует в формировании его страниц. Он запускается отдельно и выполняет служебные функции.

Это могут, например, быть парсеры, скрипты обмена информацией, периодической обработки информации и т.п.

Написание таких скриптов мало отличается от создания основного приложения. Основной исполняемый файл выглядит практически также за исключением команды инициации ядра. Она здесь выглядит так:

```
$core->Init(true);
```

Параметр этого метода определяет принадлежность файла к скрипту и отключает обработку URL. Таким образом становятся доступны GET запросы. При этом все остальные возможности IPLF остаются доступными.

Описание файлов настройки

Настройка FrameWork и построенного на нем сайта производится путем определения констант и переменных в следующих двух файлах:

Файл `settings.php`

Этот файл должен быть расположен в корневой директории сайта и содержит следующие параметры (константы):

SCRIPT_FOLDER - Директория установки скриптов относительно корневой домена. Если это корневая директория, то указывается "/", если внутренняя, то "/folder/" или "/folder/subfolder/".

WEB_PROTOCOL - web протокол, например, "http://" или "https://"

REF_END - Окончание ссылки (например, "/", ".html")

Настройки базы данных:

DB_HOST - хост базы

DB_USER - пользователь

DB_PASS - пароль пользователя

DB_BASE - имя базы данных

DB_PREFIX - префикс таблиц

SECURITY_KEY - Ключ безопасности сайта. Используется для шифрования конфиденциальных данных.

Кодировки:

DOCUMENT_CHARSET - в виде "windows-1251", "utf-8" и т.д.

DB_CHARSET - в виде "cp1251", "utf8" и т.д.

COLLATION_CHARSET – в виде "cp1251_general_ci", "utf8-unicode_ci" ит.д.

Файл `config.php`

Этот файл располагается в директории FrameWork и содержит следующие параметры:

Массивы, определяющие области доступа*.

\$wfconfig_access_areas - массив определяющий особые области доступа и разделы, в которых они расположены. Область public здесь не определяется.

\$wfconfig_access_areas_ids – массив, определяющий идентификаторы областей доступа, включая область public.

\$wfconfig_supported_get - массив определяющий какие GET запросы не будут экранироваться в ссылке. Любая входящая на сайт ссылка обрабатывается и любой GET запрос экранируется слешем (либо другими заданным в настройках символами) в конце и не воспринимается как запрос. Это делается по соображениям безопасности и

особенностей адресации. Этот массив содержит части запросов, которые не должны экранироваться. Пример:

```
$wfconfig_supported_get = Array(
    '0' => '&utm_'
);
```

Константы:

WFCONFIG_ARCHITECTURE – тип архитектуры IPLF*

WFCONFIG_COMPONENTS_FOLDER – директория расположения компонентов

WFCONFIG_VIEW_FOLDER – директория расположения шаблонов

WFCONFIG_COMPONENTS_URI – путь url директории компонентов (от корневой директории сайта без последнего /, например, 'components')

WFCONFIG_VIEW_URI – путь url директории шаблонов (от корневой директории сайта без последнего /, например, 'view')

Константы использования стандартных SQL таблиц IPLF **

WFCONFIG_USE_ALIASES_TABLE – использовать таблицу псевдонимов ссылок.

WFCONFIG_GET_SETTINGS_TABLE – считывать таблицу установок

WFCONFIG_USE_SECTIONS_TABLE – использовать для построения кэша компонентов стандартную таблицу

Регистрация и пользователи

WFCONFIG_INITIATE_REGISTRED_USER – проверять вошедшего пользователя на регистрацию в соответствии со стандартной таблицей пользователей

WFCONFIG_USE_COOKIE_AUTHORIZATION – использовать авторизацию через cookie. Используется стандартная таблица сессий**.

WFCONFIG_SESSION_TIME – время жизни сессии при использовании авторизации через cookie

WFCONFIG_USER_AVATAR_FOLDER – директория хранения аватаров

WFCONFIG_USER_AVATAR_EXTENSION – расширение аватаров

WFCONFIG_USER_PASS_FIELD – название поля пароля пользователя в таблице пользователей ***

WFCONFIG_USER_KEY_FIELD - название поля ключа пользователя в таблице пользователей ***

* - подробно описано в разделе «Основы разработки приложений».

** - стандартные таблицы IPLF описаны в соответствующем разделе.

*** - для повышения взломобезопасности сайта названия этих полей не фиксировано.

Базовые классы (файл base.classes.php)

Данный файл содержит базовые классы для формирования страниц сайта при помощи IPLF.

Класс FRAME_CORE

Основной класс, устанавливающий основные настройки и параметры сайта, обрабатывающий ссылку на страницу, а также глобальные объекты.

Класс также предназначен для окончательного формирования html кода страницы при помощи получения информации из скриптов текущего компонента.

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|---------------------|---|--------------|
| str \$component | Идентификатор текущего компонента | |
| str \$content | Сформированный компонентом html код | |
| str \$tpl | Объект шаблона лэяута | |
| str \$access_folder | Директория области доступа | |
| array \$parameters | Массив параметров ядра (работать с его значениями можно как напрямую, так и с помощью методов FRAME_CORE::Parameter() и FRAME_CORE::SetParameter()) | |

Методы класса

| | | |
|---|--|-------|
| static obj getInstance() | | |
| Получение существующего объекта класса | | |
| bool Init([\$script]) | | |
| Инициации профиля страницы и получение основных данных. | | |
| bool \$script | Параметр, определяющий входит ли данная страница в основной массив данных сайта или является одиночным скриптом и не должна обрабатываться как основная. Например, может использоваться при создании скриптов планировщика заданий или служебных скриптов. | false |

| | | |
|---|---|--------------------|
| bool DBInit([\$script]) | | |
| Подключает основную базу данных | | |
| protected bool DefineViewConstants() | | |
| Иницирует определение констант, которые содержат местоположение файлов шаблонов. | | |
| static str Parameter(\$parameter[, \$key]) | | |
| Получение параметров сайта. | | |
| str \$parameter | Получаемый параметр | |
| str \$key | В случае, если получаемый параметр является массивом, то в этой переменной указывается ключ нужного элемента. | "" (пустая строка) |
| static str UV(\$num)* | | |
| Получение значения элемента массива параметра \$url_variables ядра. Альтернатива выполнения метода FRAME_CORE::Parameter('url_variables', \$num). Делает код более компактным при частом использовании параметра \$url_variables. | | |
| int \$num | Ключ элемента массива | |
| static str SetParameter(\$val, \$parameter[, \$key[, \$add]]) | | |
| Определение параметра сайта (для текущего представления) | | |
| mixed \$val | Значение параметра | |
| str \$parameter | Определяемый параметр | |
| str \$key | В случае, если определяемый параметр является массивом, то в этой переменной указывается ключ нужного элемента. | "" (пустая строка) |
| \$add | Для строковых параметров определяет заменить его или добавить значение в конец строки | false |
| static str AccessArea() | | |
| Возвращает идентификатор текущей области доступа. | | |
| static str AccessAreaID() | | |
| Возвращает ID текущей области доступа. | | |
| static str Section() | | |

| |
|---|
| Возвращает идентификатор текущего раздела. |
| static obj DB() |
| Возвращает объект базы данных. |
| static obj User() |
| Возвращает объект зарегистрированного в данный момент пользователя. |
| bool RunController() |
| Метод инициации компонента (раздела) для получения html кода. |
| bool LayoutBuild() |
| Получение шаблона лэйаута и его обработка. |
| str Render() |
| Получение окончательного html кода. |
| static Error404() |
| Переадресация на страницу 404 ошибки. |
| str Execute() |
| Последовательное выполнение методов FRAME_CORE::RunComponent(), FRAME_CORE::LayoutBuild() и FRAME_CORE::Render() как самой распространенной последовательности методов класса для упрощения кода. |

* - Доступны с версии файла 1.1.

Класс CONTROLLER

Данный класс предназначен для организации доступа к компонентам, а также является родительским для класса контроллера компонента.

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|-------------------|------------------------------------|--------------------|
| str \$layout | Шаблон лэйаута текущего компонента | "general.tpl" |
| str \$content | Формируемый компонентом html код | "" (пустая строка) |
| str \$title | Значение метатега title | "" (пустая строка) |
| str \$description | Значение метатега description | "" (пустая строка) |
| str \$keywords | Значение метатега keywords | "" (пустая строка) |

Методы класса

| | | |
|--|------------------------|--|
| static bool Initiate(\$section) | | |
| Инициация раздела компонента по его идентификатору без указания схемы архитектуры. | | |
| str \$section | Идентификатор раздела. | |
| static bool PFCSInitiate(\$section) | | |
| Инициация раздела компонента по схеме PFCS. | | |
| str \$section | Идентификатор раздела. | |
| static bool PFSInitiate(\$section) | | |
| Инициация раздела по схеме PFS. | | |
| str \$section | Идентификатор раздела. | |
| static bool ReadComponentsCache() | | |
| Получение массива компонентов и разделов из файла кэша компонентов. | | |
| static bool CacheComponents() | | |
| Создание файла кэша структуры компонентов. | | |
| static str CacheFromTables() | | |
| Получение сериализованного массива структуры компонентов из стандартных таблиц. | | |
| static str CacheFromFiles() | | |
| Получение сериализованного массива структуры компонентов из файлов init.php компонентов. | | |

Работа с шаблонами (файл `template.classes.php`)

Шаблоны являются неотъемлемой частью любого проекта на основе системы, использующей MVC модель проектирования, в том числе и IPLF.

Шаблоны IPLF достаточно просты, но при этом обладают высокой гибкостью.

Файл шаблона может быть простым, то есть состоящим из единого блока и составным. Составные шаблоны разбиты на части (фрагменты), которые обрабатываются отдельно и могут быть подставлены друг в друга.

Переменные и конструкции

Переменные

Переменные шаблонов определяются в файле модели. Для вывода значения переменной в браузер шаблон должен содержать следующую конструкцию:

`<[имя_переменной]>`

Имя переменной не должно содержать пробелов и должно содержать только латинские буквы, цифры и служебные символы (- _ # * / \ ? @ и т.п.)

`<[base_url]>` - является предопределенной переменной и содержит значение константы `BASE_URL`.

Служебные конструкции

Служебными конструкциями шаблонов IPLF называют выражения, предназначенные для выполнения логических операций и вставки различных элементов в html текст.

Операция «если». Возможны следующие варианты:

`<[IF имя_переменной]>`

Текст отображаемый при условии, что значение переменной отличается от пустой строки

`<[IF имя_переменной END]>`

или

`<[IF имя_переменной условие значение]>`

Текст отображаемый при условии, что выполняется условие сопоставления значения переменной с заданным значением

`<[IF имя_переменной условие значение END]>`

Доступные значения условия:

| | |
|----|----------|
| = | равно |
| != | не равно |
| > | больше |
| < | меньше |

>= больше или равно
<= меньше или равно

Значение для сравнения не должно содержать пробелов. Также допускается заключение значения в одинарные или двойные кавычки, что полезно, например, для сравнения с пустой строкой.

Перебор значений. Конструкция выглядит следующим образом:

```
<[FOREACH имя_переменной]>  
    Текст отображаемый для каждого значения переменной  
<[FOREACH имя_переменной END]>
```

Переменная передаваемая в конструкцию должна содержать одномерный массив значений. Переменная с тем же именем используемая в теле конструкции будет содержать значение очередного элемента массива.

Вставка PHP кода. Вид конструкции:

```
<[PHP идентификатор]>  
    PHP код  
<[PHP идентификатор END]>
```

В качестве идентификатора используется любое значение, отвечающее тем же требованиям что и имя переменной. Идентификаторы конструкций всех используемых при формировании страницы шаблонов (в том числе и лэйаутов) не должны повторяться.

PHP код должен сформировать текстовую переменную \$result, значение которой при формировании html текста будет подставлено вместо обрабатываемой конструкции.

Определение фрагментов

Для определения фрагментов шаблона используется следующая конструкция:

```
##SET_FRAGMENT идентификатор_фрагмента##  
    Тело фрагмента  
##SET_FRAGMENT идентификатор_фрагмента END##
```

Идентификатор раздела отвечает тем же требованиям что и имя переменной и должен быть уникальным для обрабатываемого шаблона.

Класс VIEW

Класс для хранения данных шаблона, их обработки и формирования результирующего html кода.

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|---------------------------|------------------------|--------------|
| protected str \$from_tpl | Содержит текст шаблона | |
| protected array \$com_var | | |

Методы класса

| | | |
|---|--|--------------------|
| bool __construct([\$file]) | | |
| Конструктор класса | | |
| str \$file | Имя файла шаблона. Путь указывается от директории видов (VIEW_PATH). Если этот параметр указан, то текст шаблона автоматически загружается из указанного файла. Если не указан, то необходимо вызвать один из двух методов, описанных ниже | "" (пустая строка) |
| bool CostructFromFile(\$file) | | |
| Метод получения текста шаблона из файла | | |
| str \$file | Имя файла шаблона. Путь указывается от директории видов (VIEW_PATH). | |
| bool CostructFromString(\$text) | | |
| Метод получения текста шаблона из текстовой переменной | | |
| str \$text | Текст шаблона | |
| str GetGeneral() | | |
| Получение результирующего html кода простого шаблона. | | |
| str GetFragment(\$fragment) | | |
| Получение результирующего html кода фрагмента составного шаблона. | | |
| str \$fragment | Идентификатор получаемого фрагмента | |
| bool SetVar(\$variable, \$value) | | |
| Устанавливает значение переменной. | | |
| str \$variable | Имя переменной | |
| str \$value | Значение переменной | |

| | | |
|--|--|--|
| bool ClearComVars() | | |
| Очищает массив переменных шаблона | | |
| protected str PutComVars(\$text) | | |
| Заменяет переменные шаблона их значениями. | | |
| str \$text | Текст шаблона на данной стадии обработки | |

Пример

Простой шаблон

Для примера пусть это будет лэйаут, с именем template1.tpl. Надо отметить, что и шаблоны и лэйауты могут быть как простыми, так и составными.

```
<html>
<head>
    <title>Тест шаблонов</title>
</head>
<body>

<h1>Демонстрация работы шаблонов страницы</h1>

<[images]>

<[IF sky]>Сообщение: <[sky]><[IF sky END]>

<[PHP code1]>
    $result = '';
    if ( 2>1 ) { $result = '<p>Два больше одного</p>'; }
<[PHP code1 END]>

</body>
</html>
```

Составной шаблон

Для примера пусть это будет шаблон, с именем template2.tpl.

```
##SET_FRAGMENT img_result##
Наши изображения:<br />
<[images]>
<p>
<[IF count != "">Изображения есть. <[IF count != "" END]>
```

```

<[IF count > 0]>Всего <[count]> изображений. <[IF count > 0 END]>
</p>
##SET_FRAGMENT img_result END##

##SET_FRAGMENT imgs##
<[FOREACH image]>
    <div></div>
<[FOREACH image END]>
##SET_FRAGMENT imgs END##

```

Обработчик модели

```

<?

// переменные
$images = Array("img/1.jpg", "img/2.jpg", "img/3.jpg");
$sky = 'Небо голубое';

// подключаем шаблоны (второй шаблон для примера подключаем в 2 шага)
$t1 = new VIEW('layouts'.DIRECTORY_SEPARATOR.'template1.tpl');
$t2 = new VIEW();
$t2->ConstructFromFile('templates'.DIRECTORY_SEPARATOR.'template2.tpl');

// устанавливаем значения переменных изображений
$t2->SetVar('image',$images);
$count = count($images);
if ( $count>0 ) { $t2->SetVar('count',$count); } else { $t2->
SetVar('count',''); }

// получаем список изображений и устанавливаем его как значение переменной
$imgs = $t2->GetFragment('imgs');
$t2->SetVar('images',$imgs);

// получаем результат списка
$img_result = $t2->GetFragment('img_result');

// устанавливаем значения переменных лэйаута
$t1->SetVar('images',$img_result);
$t1->SetVar('sky',$sky);

// выводим результат обработки шаблонов
echo $t1->GetGeneral();

?>

```

Код результата выполнения

```
<html>
<head>
  <title>Тест шаблонов</title>
</head>
<body>

<h1>Демонстрация работы шаблонов страницы</h1>

Наши изображения:<br />
  <div></div>
  <div></div>
  <div></div>
<p>
Изображения есть.
Всего 3 изображений.
</p>

Сообщение: Небо голубое

<p>Два больше одного</p>

</body>

</html>
```

Инструменты работы с базами данных (файл db.classes.php)

IPLF предоставляет разработчикам набор инструментов для взаимодействия интернет приложений с базами данных SQL.

Данный файл содержит только один класс.

Класс DB

Данный класс предназначен для работы с базами данных. Он может:

- ✓ Создавать соединения с базами данных
- ✓ Выполнять SQL запросы
- ✓ Обрабатывать и получать данные полученные в результате выполнения запроса

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|--------------------------|---|----------------------------|
| str \$link_error_include | Путь к файлу сообщения об ошибке соединения | BASE_PATH.'/sqlerror.php' |
| bool \$log_errors | Логировать ли ошибки | true |
| str \$error_prefix | Начальная фраза сообщения об ошибке | “Не могу выполнить запрос” |

Методы класса

| bool __construct([\$main[, \$error_file]]) | | |
|--|--|--------------------|
| Конструктор класса | | |
| bool \$main | Указывает, является ли соединение основным для сайта или дополнительным. Если соединение основное, то в качестве параметров для соединения используются значения по умолчанию, назначенные в файле настроек. Если же соединение дополнительное, то метод DB::Connect запускается отдельно с прямым указанием параметров соединения | true |
| str \$error_file | то же самое, что общий параметр \$link_error_include. Позволяет указать файл сообщения об ошибке соединения | “” (пустая строка) |

| | | |
|---|--|---------|
| | непосредственно при создании объекта класса | |
| bool Connect([\$db_host[, \$db_user[, \$db_pass[, \$db_base]]]]) | | |
| Метод для создания соединения с базой данных в качестве входящих параметров по умолчанию используются константы, заданные в файле настройки settings.php (DB_HOST, DB_USER, DB_PASS, DB_BASE соответственно) | | |
| str \$db_host | Имя хоста базы данных | DB_HOST |
| str \$db_user | Имя пользователя | DB_USER |
| str \$db_pass | Пароль пользователя | DB_PASS |
| str \$db_base | Имя базы данных | DB_BASE |
| array GetFirstResult(\$query) | | |
| Возвращает первую строку результата запроса в виде одномерного массива. В случае, если не найдено ни одной записи возвращает false. | | |
| str \$query | sql запрос | |
| array GetResult(\$query) | | |
| Возвращает все строки результата запроса в виде двумерного массива. В случае, если не найдено ни одной записи возвращает false. | | |
| str \$query | sql запрос | |
| int RowsCount(\$query) | | |
| Возвращает количество строк, полученных в результате выполнения sql запроса. | | |
| str \$query | sql запрос | |
| int RowsCountByConditions(\$table, \$conditions) | | |
| Возвращает количество строк таблицы, соответствующих условиям. В случае, если не найдено ни одной записи возвращает false. Отличается от метода DB::RowsCount использованием меньшего количества аппаратных ресурсов. | | |
| str \$table | Таблица, в которой производится поиск | |
| str \$conditions | условия поиска записей в формате sql. Например, "WHERE `type`=3 LIMIT 10,10" | |
| bool Go(\$query) | | |
| Метод для выполнения запроса не требующего возврата значений (например, INSERT, DELETE, ALTER и т.п.). Возвращает true, если запрос выполнен успешно. | | |
| str \$query | sql запрос | |

| | | |
|--|-----------------------------|--|
| resource GetSource(\$query) | | |
| Возвращает результат выполнения sql запроса в необработанном виде (идентификатор результата), который в дальнейшем может быть обработан с помощью стандартных функций php для работы с sql. | | |
| str \$query | sql запрос | |
| bool MultyGo(\$query) | | |
| Метод для выполнения нескольких запросов не требующих возврата значений (например, INSERT, DELETE, ALTER и т.п.). Запросы должны быть разделены двойной точкой с запятой (;). Возвращает true, если запрос выполнен успешно. | | |
| str \$query | sql запрос | |
| bool CheckForErrors(\$query) | | |
| Метод, проверяющий sql запрос на наличие ошибок. Возвращает true, если ошибок не обнаружено и false, если выполнение запроса возвращает ошибку. | | |
| str \$query | sql запрос | |
| bool TableExists(\$table) | | |
| Метод проверяющий наличие таблицы в подключенной базе данных. Возвращает true, если таблица обнаружена и false, если нет. | | |
| str \$table | Искомая таблица базы данных | |
| bool FieldExists(\$table, \$field) | | |
| Метод проверяющий наличие поля в таблице базе данных. Возвращает true, если поле обнаружено и false, если нет. | | |
| str \$table | Таблица базы данных | |
| str \$field | Искомое поле таблицы | |

Пример

```
<?
// Создание объекта класса
// В данном случае создается подключение к базе данных
// через параметры, указанные в файле settings.php
$db = newDB();
```



```

// добавим запись
$query = "INSERT INTO `table1` VALUES ('5','some_info',1)";
$db->Go($query);

// получим уникальную запись по ID
$query = "SELECT * FROM `table2` WHERE `id` = 54";
$item = $db->GetFirstResult($query);
echo $item['name'];

// получим все записи определенного типа и выведем их ID
$query = "SELECT * FROM `table2` WHERE `type` = 3";
$items = $db->GetResult($query);
foreach ($items as $item) {
    echo $item['id'];
}

// создадим соединение по произвольным параметрам
$db1 = new DB(false);
$db1->Connect('localhost','dbuser','password','dbase');

// проверим существование в этой базе таблицы table3
if ( $db1->TableExists('table3') ) { echo 'OK'; }

// посчитаем количество записей с названием города 'Москва'
$rows = $db1->RowCountByConditions('table4', "WHERE `city`='Москва'");

?>

```

Инструменты работы с пользователями (файл `user.classes.php`)

В данном файле находятся средства создания и управления объектами пользователей, а также их авторизации и аутентификации. Большинство методов классов этого файла доступно только при использовании подключения к базе данных SQL и использовании стандартных таблиц.

Виды авторизации пользователей на сайте

IPLF предполагает два вида авторизации пользователей и хранения параметров сессий. Выбираются они исходя из задач, нагрузки сайта и параметров сервера.

Авторизация через PHP сессию (без использования cookie)

При этом виде авторизации для хранения параметров пользователя используется обычная PHP сессия.

Этот вариант более безопасен и не увеличивает нагрузку на sql сервер. Но при этом увеличивает количество операций чтения/записи на сервере, что может быть нежелательным при большом количестве посетителей. Также время жизни сессии ограничено настройками сервера.

Авторизация через cookie

Параметры сессии пользователя хранятся в базе данных сайта (см. описание стандартных страниц). В cookie браузера пользователя хранится идентификатор сессии. При авторизации, а также при дальнейших входах на сайт до истечения срока жизни cookie, создается PHP сессия с этим идентификатором, содержащая все определенные ранее и сохраненные переменные.

Такой вид авторизации не привязан к времени жизни PHP сессии. Сессия открывается только для авторизованных пользователей, что важно для высоконагруженных проектов. Из минусов можно отметить небольшое повышение нагрузки на SQL сервер и меньшую защищенность ресурса.

Класс `USER_STATIC`

Данный класс содержит статические методы для авторизации и регистрации пользователя.

Методы класса

| |
|---|
| <code>static str PassHash(\$pass, \$user_key)</code> |
| Формирует хэш пароля пользователя. |

| | | |
|---|---|---------|
| str \$pass | Пароль | |
| str \$user_key | Индивидуальный ключ безопасности пользователя, используемый для формирования хэша | |
| static str GenerateKey() | | |
| Возвращает символьно-цифровую строку длиной 18-22 символа, которая может быть использована как ключ пользователя. | | |
| static obj Login(\$login, \$pass[, \$field]) | | |
| Авторизация пользователя. | | |
| str \$login | Введенный логин (или другой параметр авторизации) пользователя | |
| str \$pass | Пароль | |
| str \$field | Поле таблицы, используемое в качестве логина | "login" |
| static obj Logout() | | |
| Отменяет авторизацию пользователя. | | |
| static bool UpdateSessionTable() | | |
| Обновляет таблицу сессий в соответствии с текущими параметрами. Используется при авторизации с помощью cookie. | | |
| static bool StartSession(\$sid) | | |
| Открывает сессию и записывает в нее все необходимые параметры если пользователь авторизован. Используется при авторизации с помощью cookie. | | |
| str \$sid | Идентификатор сессии | |

Класс USER

Данный класс предназначен для работы создания объекта пользователя и работы с ним. Используется только при условии использования стандартных таблиц IPLF.

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|----------|-----------------|--------------|
| int \$id | ID пользователя | |

| | | |
|-------------------------|---|------|
| str \$name | Имя пользователя | |
| str \$login | Логин пользователя | |
| str \$email | E-mail пользователя | |
| str \$group_id | ID группы пользователя | 0 |
| str \$group | Идентификатор группы пользователя | |
| str \$group_name | Название группы пользователя | |
| bool \$group_active | Активность группы пользователя | |
| str \$group_description | Описание группы пользователя | |
| bool \$area_access | Доступна ли текущая область доступа | true |
| int \$section_access | Уровень доступа к текущему разделу | 0 |
| str \$avatar | Полный адрес файла аватара | |
| str \$registration_date | Дата и время регистрации в формате unix | |
| array \$parameters | Массив дополнительных параметров | |

Методы класса

| | | |
|--|--|---|
| bool ConstructFromTable(\$ident[, \$mode]) | | |
| Получение свойств пользователя для создания объекта из таблицы | | |
| str \$ident | Строка, используемая как идентификатор при поиске записи пользователя (ID или логин) | |
| str \$mode | Режим поиска. i - поиск по ID, l - поиск по логину | i |
| bool ConstructFromArray(\$array) | | |
| Создание объекта пользователя из массива данных | | |
| array \$array | Массив свойств пользователя | |
| bool AddParameter(\$name, \$value) | | |
| Добавление элемента в массив свойств \$parameters объекта класса | | |
| str \$name | Идентификатор свойства (ключ массива) | |

| | | |
|---|--------------------|--|
| str \$value | Значение параметра | |
| bool GetGroup() | | |
| Получение свойств группы пользователя. Заполняет параметры \$group, \$group_name, \$group_active, \$group_description объекта данного класса. | | |
| bool GetAccess() | | |
| Получение параметров доступа пользователя к текущей области и разделу. Заполняет параметры \$area_access и \$section_access объекта класса. | | |

Примеры

Пример 1

Создание объекта пользователя и операции с ним

```
<?

// создаем пользователя по ID
$user = new USER();
$user->ConstructFromTable('597');

// выводим его имя
echo $user->name;

// проверяем доступ пользователя к текущему разделу
$user->GetAccess();
if ( $user->section_access==0 ) { echo 'Доступ запрещен'; }

// выводим название группы пользователя
$user->GetGroup();
echo $user->group_name;

?>
```

Пример 2

Вывод всех пользователей группы и создание массива их объектов

```
<?

// объявление массива пользователей
$users = Array();

// получаем из таблицы пользователей нужные данные
```

```
$query = "SELECT * FROM `table2` WHERE `type` = 3";
$items = $db->GetResult($query);

// создание объектов и вывод
foreach ($items as $item) {
    $user = new USER();
    $user->ConstructFromArray($item);
    echo '['.$user->id.'] '.$user->login.'<br />';
    $users[] = $user;
}

?>
```

Инструменты работы с формами (файл form.classes.php)

IPLF предоставляет разработчикам набор инструментов для создания и обработки данных html форм.

Работа ведется путем создания объекта формы, содержащего экземпляры полей. Полям можно задавать значения (как вручную, так и из запроса), а также получать их значение. Полям можно устанавливать любые атрибуты для получения html кода формы.

Класс FORM

Данный класс предназначен для работы создания объекта пользователя и работы с ним. Используется только при условии использования стандартных таблиц IPLF.

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|---------------------|---|-----------------------|
| str \$template_type | Тип шаблона формы. При значении "text" шаблон передается текстовой строкой. Также возможно значение "file". В этом случае текст шаблона берется из указанного файла | "text" |
| str \$template | Шаблон формы. В зависимости от указанного типа содержит либо текст шаблона, либо имя файла шаблона с указанием полного пути к нему | |
| str \$action | Значение атрибута action формы | "/" |
| str \$method | Значение атрибута method формы | "POST" |
| str \$autocomplete | Значение атрибута autocomplete формы | "" (пустая строка) |
| str \$enctype | Значение атрибута enctype формы | "multipart/form-data" |
| str \$name | Имя формы | |
| bool \$novalidate | Использовать ли атрибут novalidate для формы | false |
| str \$target | Значение атрибута target формы | "" (пустая строка) |
| str \$doctype | Значение атрибута doctype формы | "xhtml" |

| | | |
|-----------------|---|--|
| str \$attributs | Дополнительные произвольные атрибуты тега формы | |
|-----------------|---|--|

Методы класса

| | | |
|--|---|--------------------|
| bool NewField(\$type, \$name[, \$value]) | | |
| Добавление нового поля в форму. | | |
| str \$type | Тип поля формы. Типы допустимых типов полей описаны ниже | |
| str \$name | Имя поля | |
| str \$value | Значение поля | "" (пустая строка) |
| bool SetValue(\$value[, \$name]) | | |
| Установка значения поля. | | |
| str \$value | Значение поля | |
| str \$name | Имя поля. Необязательный атрибут. В случае, если он не указан (пустая строка), атрибут присваивается последнему добавленному полю | "" (пустая строка) |
| bool AddFieldAttribut(\$attribute, \$value[, \$name]) | | |
| Добавление или изменение предустановленного атрибута поля. | | |
| str \$attribute | Название атрибута. Список атрибутов находится ниже | |
| str \$value | Значение атрибута | |
| str \$name | Имя поля. Необязательный атрибут. В случае, если он не указан (пустая строка), атрибут присваивается последнему добавленному полю | "" (пустая строка) |
| bool AddTagAttributs(\$value[, \$name]) | | |
| Добавление или изменение произвольных атрибутов поля формы. | | |
| str \$value | Атрибуты тега | |
| str \$name | Имя поля. Необязательный атрибут. В случае, если он не указан (пустая строка), атрибут присваивается последнему добавленному полю | "" (пустая строка) |

| | | |
|---|---|--------------------|
| bool AddOption(\$name, \$value, \$text[, \$selected[, \$disabled[, \$label]]) | | |
| Добавление тегов option для полей типа select и datalist. | | |
| str \$name | Имя поля | |
| str \$value | Значение тега option | |
| str \$text | Текст внутри тега | |
| bool \$selected | Добавить атрибут selected | false |
| bool \$disabled | Добавить атрибут disabled | false |
| str \$label | Значение атрибута label | "" (пустая строка) |
| bool ChangeOptionAttrib(\$name, \$ident, \$attribute, \$value) | | |
| Изменение атрибутов selected, disabled и label тегов option для полей типа select и datalist. | | |
| str \$name | Имя поля | |
| str \$ident | Идентификатор тега. В качестве идентификатора указывается значение атрибута value элемента option | |
| str \$attribute | Название атрибута | |
| str \$value | Значение атрибута | |
| bool AddValue(\$value, \$name) | | |
| Добавление нового значения для группы radio кнопок. | | |
| str \$value | Значение кнопки. | |
| str \$name | Имя поля. Необязательный атрибут. В случае, если он не указан (пустая строка), атрибут присваивается последнему добавленному полю | "" (пустая строка) |
| bool GetRequestValues() | | |
| Получение и обработка данных запроса. Значения полей заполняются данными полученными из запроса. Важно! Кавычки в получаемых данных экранируются. | | |
| bool UnSlash([\$name]) | | |
| Удаляет экранирование из значений полей. | | |

| | | |
|---|--|--------------------|
| str \$name | Имя поля. В случае, если он не указан (пустая строка), обрабатывается значение только этого поля | "" (пустая строка) |
| str GetFieldValue(\$name) | | |
| Получение значения поля. | | |
| str \$name | Имя поля | |
| array GetFieldsValues() | | |
| Получение массива значений полей формы. Ключами массива являются имена полей. | | |
| str Render() | | |
| Получение html текста формы. | | |

Типы полей и атрибуты

Класс FORM поддерживает следующие типы полей:

"select", "textarea", "hidden", "text", "password", "file", "button", "submit", "reset", "image", "radio", "checkbox", "datalist", "number", "range", "date", "datetime", "datetime-local", "time", "email", "search", "url", "tel", "color", "month", "week"

Все типы полей кроме "select" и "textarea" представляют собой виды тега input.

Список атрибутов и параметров полей:

Для всех полей:

Стандартные атрибуты:

str form, name, id, value, doctype, tabindex, accesskey

bool disabled, autofocus, required

Дополнительные параметры:

str attributs - любые неописанные атрибуты в произвольной форме.

bool is_html - используется для текстовых полей и определяет, возможен ли ввод html тегов.

Для полей textarea

bool readonly

str maxlength, placeholder, cols, rows, wrap

Для полей text

bool readonly

str maxlength, placeholder, autocomplete, pattern, size

Для полей password

bool readonly

str maxlength, placeholder, autocomplete, size

Для полей select

bool multiple

str size

Для полей file

str accept

bool multiple

Для полей submit

str formaction, formenctype, formmethod, formtarget

bool formnovalidate

Для полей radio, checkbox

bool checked

Для полей search, url, tel

str autocomplete, pattern

Для полей email

str autocomplete, pattern

bool multiple

Для полей number, range

str max, min, step

Для полей date

str max, min

Для полей image

str align, alt, border, src

Шаблоны форм

Принцип создания шаблонов форм ничем не отличается от общего для IPLF принципа.

В качестве шаблона передается простой шаблон (не разделенный на фрагменты). Поэтому стоит либо задавать файл простого шаблона, либо передавать фрагмент составного шаблона в виде текста.

Логические операции и коды здесь не выполняются без дальнейшей обработки шаблона.

Шаблон должен содержать код внутри тега <form>. Сам тэг добавится при построении результирующего кода.

Поля формы вставляются в шаблон в виде обычных переменных шаблона. Исключение составляют только radio кнопки. Они вставляются при помощи конструкции

<[имя_поля||значение]>

Здесь имя поля – имя элемента radio объекта формы (атрибут name группы кнопок), а значение – атрибут value тега кнопки.

Пример

Создадим форму авторизации.

Построение формы, обработка данных и вывод html кода.

```
<?

// данные для входа
$login = 'user';
$pass = '1234';
$error = false;

// создаем объект формы
$form = new FORM();

// создаем поля
$form->NewField('text','login');
$form->NewField('password','pass');
$form->NewField('radio','isuser');
// добавляем значение последнему созданному элементу
$form->AddValue('no');
$form->NewField('checkbox','agreement');
// добавляем значение элементу isuser
$form->AddValue('yes','isuser');

// проверяем отправку данных
if (isset($_POST['action'])) {
    // отправка была
    // получаем данные запроса
    $form->GetRequestValues();
    $posted = $form->GetFieldsValues();
    // проверяем на соответствие
    if ( $posted['login']!=$login ) { $error = true; }
```

}

[illegible]

Внешний вид формы

Форма входа

Логин

Пароль

Вы пользователь? ☒ Нет ☐ Да

☒ Я соглашаюсь с условиями

Инструменты работы с изображениями (файл `images.classes.php`)

Работа с изображениями в IPLF основана на PHP библиотеке GD, но при этом значительно упрощает ее использование. Предназначен для загрузки изображений на сервер, создания графиков и диаграмм, а также любых других действий с графикой.

Класс **IMAGE**

При помощи этого класса Вы можете:

- ✓ Создавать копии изображений на сервере, вписанные в определенные размеры или растянутые по ним
- ✓ Создавать копии изображений на сервере, отмасштабированные по заданному коэффициенту
- ✓ Накладывать "водяные знаки" на изображения
- ✓ Изменять формат изображений
- ✓ Создавать динамические графические изображения
- ✓ Сохранять изображения на сервере
- ✓ Выводить полученные изображения в браузер

Класс оперирует тремя основными объектами:

Source - объект исходного изображения. Создается либо из существующего изображения как его копия, либо как пустой лист, на котором в дальнейшем с помощью средств библиотеки GD можно создать любое изображение.

Filigree - объект содержащий изображение водяного знака.

Result - результирующее изображение, которое формируется исходя из переданных в него параметров. Это изображение сохраняется на сервер или выводится непосредственно в браузер.

Методы класса

| bool CreateFromFile(\$filename) | | |
|--|---|--|
| Создает исходное изображение из файла на сервере. Если файл нечитаемый или в неверном формате, то возвращает false. | | |
| str \$filename | Имя файла изображения с абсолютным путем до него. Поддерживаются изображения jpg, png и gif | |
| bool CreateBlank(\$width, \$height) | | |
| Создает исходное изображение как пустой лист. Далее с ним можно работать с помощью графических функций этого класса или стандартных средств библиотеки GD. | | |

| | | |
|---|---|--|
| int \$width | Ширина листа | |
| int \$height | Высота листа | |
| obj GetWorkObject() | | |
| Возвращает созданный ранее методами IMAGE::CreateFromFile или IMAGE::CreateBlank объект изображения, либо false, если объект не был создан. Далее с ним можно работать с помощью стандартных средств библиотеки GD. | | |
| int GetSourceWidth() | | |
| Возвращает ширину исходного изображения в пикселях, либо false, если объект не был создан. | | |
| int GetSourceHeight() | | |
| Возвращает высоту исходного изображения в пикселях, либо false, если объект не был создан. | | |
| bool SetResizeMethod(\$mode) | | |
| Устанавливает режим масштабирования изображения. Используется вместе с методом IMAGE::SetSizes или IMAGE::SetRatio. В зависимости от установки параметра исходное изображение может быть либо вписано в указанный прямоугольник, либо растянуто по его размерам, либо отмасштабировано на четко указанный коэффициент. По умолчанию имеет значение "enclosed" для изображения из файла и "direct_ratio" для чистого бланка. | | |
| str \$mode | Режим масштабирования: "enclosed" - вписать в прямоугольник, "resize" - растянуть или сжать стороны до указанных размеров без сохранения пропорций, "direct_ratio" - масштабировать по коэффициенту | |
| bool SetRatio(\$ratio) | | |
| Устанавливает коэффициент при использовании режима масштабирования "direct_ratio". | | |
| float \$ratio | Коэффициент. Должен быть больше нуля. По умолчанию установлено 1 | |
| bool SetSizes(\$width, \$height) | | |
| Устанавливает коэффициент при использовании режима масштабирования "enclosed" и "resize". | | |
| int \$width | Ширина прямоугольника в пикселях. Должна быть больше нуля | |
| int \$height | Высота прямоугольника в пикселях. Должна быть больше нуля | |

| | | |
|--|---|--------|
| bool SetResultExtension(\$ext) | | |
| Устанавливает формат результирующего файла "png", "jpg", "gif". По умолчанию установлено "png". | | |
| str \$ext | Тип файла. Может принимать значения "png", "jpg", "gif" | |
| bool AddFiligree(\$filename, \$xmargin, \$ymargin[, \$xmargin_pos[, \$ymargin_pos[, \$transparency]]) | | |
| Добавляет "водяной знак" к результирующему изображению. | | |
| str \$filename | Имя файла изображения с абсолютным путем до него. Воспринимаются изображения jpg, png и gif | |
| int \$xmargin | Отступ по оси X | |
| int \$ymargin | Отступ по оси Y | |
| \$xmargin_pos | Выбирает направление отступа по оси X. Может иметь значения "left" (отступ от левого края) и "right" (отступ от правого края) | 'left' |
| \$ymargin_pos | Выбирает направление отступа по оси Y. Может иметь значения "top" (отступ от верхнего края) и "bottom" (отступ от нижнего края) | 'top' |
| \$transparency | Прозрачность водяного знака в процентах. 0 - не отображается, 100 - отображается без прозрачности. | 30 |
| bool AntiAliasing(\$enabled)* | | |
| Устанавливает режим сглаживания линий рисунка. Изначально сглаживание включено. | | |
| bool \$enabled | Состояние сглаживания: true - сглаживать, false - нет | |
| obj Point(\$x, \$y)* | | |
| Возвращает точку, которая в дальнейшем может быть использована для построения геометрических фигур. | | |
| float \$x | Расстояние от верхнего края листа в пикселях. Округляется до целого. | |
| float \$y | Расстояние от левого края листа в пикселях. Округляется до целого. | |
| bool Fill(\$color)* | | |

| | | |
|--|-------------------------------------|-------|
| Заливает всю площадь листа указанным цветом. | | |
| str \$color | HEX код цвета | |
| bool Pixel(\$color, \$point)* | | |
| Рисование точки (закрашивание одного пикселя). | | |
| str \$color | HEX код цвета | |
| obj \$point | Точка | |
| bool Line(\$color, \$point1, \$point2[, \$thickness])* | | |
| Рисование линии. | | |
| str \$color | HEX код цвета | |
| obj \$point1 | Первая точка | |
| obj \$point2 | Вторая точка | |
| int \$thickness | Толщина линии в пикселях | 1 |
| bool Rectangle(\$color, \$point1, \$point2[, \$fill])* | | |
| Рисование прямоугольника. | | |
| str \$color | HEX код цвета | |
| obj \$point1 | Первая точка (угол) | |
| obj \$point2 | Вторая точка (противоположный угол) | |
| bool \$fill | Закрашивать полностью | false |
| bool Polygon(\$color, \$points[, \$fill])* | | |
| Рисование многоугольника. | | |
| str \$color | HEX код цвета | |
| array \$points | Массив точек вершин многоугольника | |
| bool \$fill | Закрашивать полностью | false |
| bool Ellipse(\$color, \$center, \$width[, \$height[, \$fill]])* | | |
| Рисование эллипса (круга). | | |
| str \$color | HEX код цвета | |
| obj \$center | Точка центра эллипса | |
| int \$width | Ширина в пикселях по горизонтали | |

| | | |
|--|---|-------|
| int \$height | Высота в пикселях по вертикали. Если равна нулю, то рисуется круг | 0 |
| bool \$fill | Закрашивать полностью | false |
| bool Arc(\$color, \$center, \$start, \$end, \$width[, \$height[, \$fill]])* | | |
| Рисование дуги окружности или эллипса. | | |
| str \$color | HEX код цвета | |
| obj \$center | Точка центра эллипса | |
| int \$start | Угол начала дуги. 0-360 от оси x против часовой стрелки | |
| int \$end | Угол конца дуги. 0-360 от оси x против часовой стрелки | |
| int \$width | Ширина в пикселях по горизонтали | |
| int \$height | Высота в пикселях по вертикали. Если равна нулю, то рисуется круг | 0 |
| bool \$fill | Закрашивать полностью | false |
| bool Sector(\$color, \$center, \$start, \$end, \$width[, \$height[, \$fill]])* | | |
| Рисование сектора окружности или эллипса. | | |
| str \$color | HEX код цвета | |
| obj \$center | Точка центра эллипса | |
| int \$start | Угол начала сектора. 0-360 от оси x против часовой стрелки | |
| int \$end | Угол конца сектора. 0-360 от оси x против часовой стрелки | |
| int \$width | Ширина в пикселях по горизонтали | |
| int \$height | Высота в пикселях по вертикали. Если равна нулю, то рисуется круг | 0 |
| bool \$fill | Закрашивать полностью | false |
| bool Text(\$text, \$color, \$point, \$font[, \$size[, \$angle[, \$transparency]])*) | | |
| Нанесение текстовой строки на лист. | | |
| str \$text | Текстовая строка в кодировке UTF-8 | |
| str \$color | HEX код цвета | |

| | | |
|--|---|-------|
| obj \$point | Точка левого верхнего угла надписи | |
| str \$font | Имя файла и путь к используемому TrueType-шрифту (ttf) | |
| int \$size | Размер шрифта в пикселях | 14 |
| int \$angle | Угол поворота надписи в градусах. 0-360 от оси x против часовой стрелки | 0 |
| int \$transparency | Прозрачность. 0-100, при 0 видна полностью | 0 |
| bool FillContour(\$color, \$contour_color, \$point)* | | |
| Закрашивание области ограниченной контуром определенного цвета. | | |
| str \$color | HEX код цвета, которым будет закрашена область | |
| str \$contour_color | HEX код цвета ограничивающего контура | |
| obj \$point | Любая точка внутри контура | |
| bool SaveImage(\$filename) | | |
| Сохраняет результирующее изображение на сервер по указанному адресу и имени. | | |
| str \$filename | Имя файла изображения, куда будет сохранен результат с абсолютным путем до него | |
| bool ShowImage() | | |
| <p>Выводит результирующее изображение в браузер. При использовании этого метода нужно помнить, что изображение — это бинарный файл, а браузер работает с текстом. Поэтому для корректного отображения изображения в скрипте нужно выставить заголовок</p> <pre>header("Content-type: image/png");</pre> <p>Пример дан для png файла.</p> | | |
| bool Destroy([\$filigree]) | | |
| Очищает объект, удаляя из него все созданные внутренние объекты. При этом настройки остаются и могут быть использованы при следующем обращении. | | |
| bool \$filigree | Удалить "водяной знак" | false |

* - Доступны с версии файла 1.1.

Примеры

Пример 1

Загрузка файла на сервер с изменением размера и добавлением “водяного знака”.

```
<?

// исходные данные
// загруженный на сервер необработанный файл
$tmp_img = BASE_PATH."temp/img.png";
// место и имя сохранения файла
$new_file = BASE_PATH."img/myimg.jpg";

// создаем объект класса из файла
$image = new IMAGE();
$image->CreateFromFile($tmp_img);

// задаем размеры результирующего изображения
// при этом помним, что по умолчанию стоит режим "enclosed"
$image->SetSizes(500, 400);

// меняем тип изображения
$image->SetResultExtension('jpg');

// добавляем "водяной знак"
$image->AddFiligree(BASE_PATH."img/filigree.png", 10, 10, 'right', 'bottom')

// сохраняем результат
$image->SaveImage($new_file);

?>
```

Пример 2

Копирование файлов с масштабированием.

```
<?

// исходные данные
$from = BASE_PATH.'from';
$to = BASE_PATH.'to';

// создаем объект класса из файла
$image = new IMAGE();

// меняем режим масштабирования и задаем условия
```

```

$image->SetResizeMethod('direct_ratio');
$image->SetRatio(0.6);

// открываем директорию для чтения и проходим по файлам
if ($dir = @opendir($from)) {
    while (($file = readdir($dir)) !== false) {

        // если удалось создать изображение из файла обрабатываем его
        if ($image->CreateFromFile($from.DIRECTORY_SEPARATOR.$file)) {

            // сохраняем обработанное изображение
            $image->SaveImage($to.DIRECTORY_SEPARATOR.$file);

            // удаляем в объекте все кроме настроек
            $image->Destroy();
        }
    }
    closedir($dir);
}

?>

```

Пример 3

Рисование геометрических фигур.

```

<?

// выводим изображение, поэтому не забываем заголовок
header( 'Content-type: image/png' );

// создаем объект класса (чистый лист)
$image = new IMAGE();
$image->CreateBlank(500,330);

// заливаем фон всего листа
$image->Fill('#aa4d4d');

// рисуем точку
$point = $image->Point(10, 320);
$image->Pixel('#6ff45e', $point);

// рисуем линию
$point1 = $image->Point(450, 10);
$point2 = $image->Point(10, 40);
$image->Line('#6ff45e', $point1, $point2);

// еще одна линия с толщиной 4
$point1 = $image->Point(10, 10);

```

```

$point2 = $image->Point(450, 60);
$image->Line('#ff932a', $point1, $point2, 4);

// прямоугольник с заливкой
$point1 = $image->Point(30, 50);
$point2 = $image->Point(90, 100);
$image->Rectangle('#fff', $point1, $point2, true);

// просто прямоугольник
$point1 = $image->Point(40, 60);
$point2 = $image->Point(110, 150);
$image->Rectangle('#5a2090', $point1, $point2);

// рисуем круг с заливкой
$center = $image->Point(250, 130);
$image->Ellipse('#38ddd7', $center, 100, 100, true);

// эллипс без заливки
$center = $image->Point(280, 150);
$image->Ellipse('#387add', $center, 200, 150);

// две дуги с заливкой и без
$center = $image->Point(250, 200);
$image->Arc('#122064', $center, -30, 60, 100);
$image->Arc('#a7ff0d', $center, 150, 300, 70, 200, true);

// два concentрических эллипса
$center = $image->Point(100, 250);
$image->Sector('#ffeb0d', $center, 67, 13, 140, 80);
$image->Sector('#0d4cff', $center, 15, 65, 140, 80, true);

// пятиугольник с вписанной звездой
$p1 = $image->Point(400, 200);
$p2 = $image->Point(410, 250);
$p3 = $image->Point(450, 250);
$p4 = $image->Point(420, 270);
$p5 = $image->Point(430, 300);
$p6 = $image->Point(400, 280);
$p7 = $image->Point(370, 300);
$p8 = $image->Point(380, 270);
$p9 = $image->Point(350, 250);
$p10 = $image->Point(390, 250);
$image->Polygon('#bbfaf2', Array($p1, $p3, $p5, $p7, $p9), true);
$image->Polygon('#ff0000', Array($p1, $p2, $p3, $p4, $p5, $p6, $p7, $p8, $p9, $p10));

// полупрозрачный текст
$point = $image->Point(60, 60);
$image->Text('Hello!!!', '#12ff00', $point, 'Candara.ttf', 48, 340, 60);

// выводим результат в браузер
$image->ShowImage();
die();

?>

```

Создание постраничного вывода (файл pagination.classes.php)

Класс PAGINATION

Используется для создания навигации при постраничном выводе информации.

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|----------------------|---|--------------|
| int \$onpage | Количество записей, выводимых на одну страницу | 20 |
| int \$page | Текущий номер страницы | 1 |
| int \$start | Количество дочерних элементов (первый уровень) | 1 |
| bool \$rel_canonical | Признак отвечающий за добавление тега <link rel="canonical" href="..." /> в блок <header> документа. Для отображения тега в шаблон документа должна быть вставлена переменная <[header_dynamic_tags]> | true |

Методы класса

| | | |
|--|---|--|
| bool SetOnPage(\$value) | | |
| Устанавливает количество элементов, выводимых на странице. | | |
| int \$value | Количество элементов на странице | |
| bool SetPType(\$value) | | |
| Устанавливает тип показа номеров страниц. | | |
| int \$value | Тип показа. По умолчанию 1. Возможные значения: 0 - выводятся все номера страниц (пример: << < 1 2 3 4 5 6 7 8 9 10 > >>), 1 - страницы выводятся тремя ограниченными блоками (пример: << < 1 2 3 ... 10 11 12 13 14 ... 26 27 28 > >>) | |
| bool SetLeftPositions(\$value) | | |
| Устанавливает количество позиций в первом (левом) блоке при типе показа 1. | | |

| | | |
|--|--|--|
| int \$value | Значение количества позиций. По умолчанию 3. Например, при установке значения 5 строка выглядит так: << < 1 2 3 4 5 ... 10 11 12 13 14 ... 26 27 28 > >> | |
| bool SetRightPositions(\$value) | | |
| Устанавливает количество позиций в третьем (правом) блоке при типе показа 1. | | |
| int \$value | Значение количества позиций. По умолчанию 3. Например, при установке значения 5 строка выглядит так: << < 1 2 3 ... 10 11 12 13 14 ... 24 25 26 27 28 > >> | |
| bool SetMiddlePositions(\$value) | | |
| Устанавливает количество позиций в среднем блоке при типе показа 1. | | |
| int \$value | Значение количества позиций. По умолчанию 5. Например при установке значения 7 строка выглядит так: << < 1 2 3 ... 9 10 11 12 13 14 15 ... 26 27 28 > >> | |
| bool SetPrevBlockTemplate(\$value) | | |
| Определяет шаблон блока, который дает возможность перейти на первую или предыдущую страницу списка и ставится перед номерами страниц. | | |
| str \$value | Строка, содержащая шаблон* | |
| bool SetNextBlockTemplate(\$value) | | |
| Определяет шаблон блока, который дает возможность перейти на последнюю или последующую страницу списка и ставится после номеров страниц. | | |
| str \$value | Строка, содержащая шаблон* | |
| bool SetPrevBlockNaTemplate(\$value) | | |
| Определяет шаблон блока, который дает возможность перейти на первую или предыдущую страницу списка и ставится перед номерами страниц, для первой страницы. | | |
| str \$value | Строка, содержащая шаблон* | |
| bool SetNextBlockNaTemplate(\$value) | | |
| Определяет шаблон блока, который дает возможность перейти на последнюю или последующую страницу списка и ставится после номеров страниц, для последней страницы. | | |
| str \$value | Строка, содержащая шаблон* | |

| | | |
|---|--|--|
| bool SetCurrentPositionTemplate(\$value) | | |
| Определяет шаблон текущего номера страницы. | | |
| str \$value | Строка, содержащая шаблон* | |
| bool SetPositionTemplate(\$value) | | |
| Определяет шаблон номера страницы. | | |
| str \$value | Строка, содержащая шаблон* | |
| bool SetSpacer(\$value) | | |
| Определяет html код промежутка между блоками при типе показа 1. | | |
| str \$value | Строка, содержащая код. По умолчанию имеет код "... " | |
| str GetPagination(\$items_num, str \$section) | | |
| Возвращает результирующий html код навигации постраничности. | | |
| int \$items_num | Общее количество элементов списка | |
| str \$section | Раздел сайта с необходимыми параметрами или псевдоним без окончания ссылки. Например, "articles", "news/125", "products/notebooks" | |

* - шаблоны, используемые если методы не вызваны, а также доступные переменные приведены ниже.

Шаблоны по умолчанию и переменные

Шаблон блока перед номерами страниц

```
<a href='<[base_url]><[sect]>p=1/' title='Первая'><<</a>>&nbsp;
<a href='<[base_url]><[sect]>p=<[prev]>/' title='Предыдущая'><</a>&nbsp;
```

Допустимые переменные

<[base_url]> - базовый адрес сайта

<[sect]> - раздел (ссылка без определения страницы)

<[prev]> - номер предыдущей страницы

Шаблон блока после номеров страниц

```
<a href='<[base_url]><[sect]>p=<[next]>/' title='Следующая'>></a>&nbsp;
<a href='<[base_url]><[sect]>p=<[pages_num]>/' title='Последняя'>>></a>
```

Допустимые переменные

<[base_url]> - базовый адрес сайта
<[sect]> - раздел (ссылка без определения страницы)
<[next]> - номер следующей страницы
<[pages_num]> - общее количество страниц

Шаблон блока перед номерами страниц (для первой страницы)

```
<< &nbsp;
```

Допустимые переменные

<[base_url]> - базовый адрес сайта
<[sect]> - раздел (ссылка без определения страницы)
<[prev]> - номер предыдущей страницы

Шаблон блока после номеров страниц (для последней страницы)

```
> >>
```

Допустимые переменные

<[base_url]> - базовый адрес сайта
<[sect]> - раздел (ссылка без определения страницы)
<[next]> - номер следующей страницы
<[pages_num]> - общее количество страниц

Шаблон номера текущей страницы

```
<b><[position]></b>&nbsp;
```

Допустимые переменные

<[position]> - текущий номер страницы

Шаблон номера текущей страницы

```
<a href='<[base_url]><[sect]>p=<[position]>'><[position]></a>&nbsp;
```

Допустимые переменные

<[base_url]> - базовый адрес сайта

<[sect]> - раздел (ссылка без определения страницы)

<[position]> - текущий номер страницы

Примеры

Пример 1

```
<?

// количество элементов
$items = 357;

// создаем объект постраничности
$pagination = new PAGINATION();

// задаем параметры
$pagination->SetOnPage(15);
$pagination->SetLeftPositions(5);
$pagination->SetRightPositions(5);
$pagination->SetMiddlePositions(7);

// получаем html код навигации и выводим
echo $pagination->GetPagination($items, 'posts/news');

// меняем тип вывода и выводим еще раз
$pagination->SetPType(0);
echo $pagination->GetPagination($items, 'posts/news');

?>
```

Результат:

<< < 1 2 3 4 5 ... 9 10 11 **12** 13 14 15 ... 20 21 22 23 24 > >>

<< < 1 2 3 4 5 6 7 8 9 10 11 **12** 13 14 15 16 17 18 19 20 21 22 23 24 > >>

Пример 2

CSS стили в примере опущены.

```
<?
// количество элементов
$items = 633;

// создаем объект постраничности
$pagination = new PAGINATION();

// задаем параметры
$pagination->SetPrevBlockTemplate("<div class='first_block'>
    <a href='<[base_url]><[sect]>p=1/' title='Первая'><<</a>>
    </div>");
$pagination->SetNextBlockTemplate("<div class='last_block'>
    <a href='<[base_url]><[sect]>p=<[pages_num]>/' title='Последняя'>>></a>
    </div>");
$pagination->SetPrevBlockNaTemplate("<div class='first_block_na'><<</div>");
$pagination->SetNextBlockNaTemplate("<div class='last_block_na'>>></div>");
$pagination->SetCurrentPositionTemplate("<div
class='current_position'><[position]></div>");
$pagination->SetPositionTemplate("<div class='position'>
    <a href='<[base_url]><[sect]>p=<[position]>'><[position]></a>
    </div>");
$pagination->SetSpacer("<div class='spacer'>...</div>");

// получаем html код навигации и выводим
echo $pagination->GetPagination($items, 'posts/news');

?>
```

Результат:

[<<](#) [1](#) [2](#) [3](#) [4](#) [5](#) ... [62](#) [63](#) [64](#) [>>](#)

Инструменты работы с древовидными структурами (файл tree.classes.php)

Данный файл предназначен для работы с наиболее распространенным видом организации хранения данных – структурой, где все данные разделены на родительские и дочерние элементы. Причем элемент может быть родительским для одного и дочерним для другого элемента. Каждый элемент может иметь только один родительский элемент и любое количество дочерних. Часто такие структуры называют «деревом».

Деревья можно разделить на однотипные (например, структура страниц сайта, где структурируются только страницы) и на двутипные «категория - позиция» (например, каталог товаров, в котором существуют структурированные разделы и входящие в них товары).

Построение объекта дерева состоит из следующих стадий:

- Создание массива элементов
- Работа с элементами дерева
- Структурирование элементов и создание единого объекта

Класс TREE NODE

Класс для создания элемента структуры дерева

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|-------------------------|---|--------------|
| str \$name | Имя элемента | |
| int \$id | ID элемента | |
| int \$parent | ID родительского элемента | |
| int \$children_num | Количество дочерних элементов | 0 |
| array \$children | Массив дочерних элементов. До преобразования в единый объект здесь хранятся только ID, после - объекты элементов | Array() |
| int \$items_num | Количество входящих позиций (для двутипного дерева) | 0 |
| int \$special_items_num | Количество входящих позиций с определенным признаком (например, новых или часто используемых) для двутипного дерева | 0 |

Класс TREE

Класс для создания объекта дерева

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|-----------------------------|---|--------------|
| str \$name | Имя корневого элемента | "root" |
| int \$id | ID корневого элемента | 0 |
| int \$children_num | Количество дочерних элементов (первый уровень) | 0 |
| array \$children | Массив дочерних элементов. После преобразования в единый объект содержит объекты элементов первого уровня | Array() |
| int \$items_num | Количество входящих позиций (для двутипного дерева) | 0 |
| int \$special_items_num | Количество входящих позиций с определенным признаком (например, новых или часто используемых) для двутипного дерева | 0 |
| str \$categories_table | Имя таблицы элементов. Указывается без префикса. | |
| bool \$count_children | Рассчитывать количество дочерних элементов при формировании из таблицы | false |
| str \$items_table | Таблица позиций. Указывается без префикса. | |
| bool \$all_items | Считать все позиции дочерней ветки * | false |
| bool \$count_special_items | Считать количество входящих позиций с определенным признаком | false |
| str \$special_items_field | Поле таблицы позиций определяющее особый признак | |
| mixed \$special_items_value | Значение поля таблицы позиций определяющее особый признак | |

* - если установлено значение false, то считается количество позиций, принадлежащих непосредственно обрабатываемому элементу, а если true, то учитываются позиции, принадлежащие элементам всей дочерней ветки.

Методы класса

| | | |
|--|---|-------|
| bool CreateEmptyTree() | | |
| Создает пустое дерево элементов | | |
| bool CreateElementsFromTable(\$id[, \$depth]) | | |
| Создает элементы из таблицы *. | | |
| int \$id | ID корневого элемента | |
| int \$depth | Глубина дерева в уровнях начиная с 1. Если установлено false, то строится вся ветвь | false |
| bool AddItemsCount() | | |
| Добавляет количество позиций в элементы *. | | |
| bool AddLine(\$line[, \$separator]) | | |
| Используется для создания дерева элементов из структур не имеющих четкого определения родительских и подчиненных элементов. Здесь структура задается именами разделов, расположенными в порядке от старшего к младшему (например, "Раздел1/Раздел2/.../РазделN"). Дерево элементов строится в соответствии с именами разделов. ID назначаются по порядку создания элементов. | | |
| str \$line | Строка с названиями разделов | |
| str \$separator | Разделитель имен разделов | "/" |
| bool AddElement(\$element_key, \$id, \$name, \$parent) | | |
| Добавить элемент в массив | | |
| str/int \$element_key | Ключ элемента в массиве | |
| int \$id | ID элемента | |
| str \$name | Название элемента | |
| int \$parent | ID родительского элемента | |
| bool DeleteElement(\$element_key) | | |
| Удалить элемент из массива | | |
| str/int \$element_key | Ключ элемента в массиве | |
| bool ConstructObject() | | |
| Формирует окончательный объект дерева | | |

* - Для работы этих скриптов необходимо, чтобы таблицы элементов содержали следующие поля: `id` - ID элемента, `parent` - ID родительского элемента, `parents_line` - строка родительских элементов, `name` - название элемента. Строка родительских элементов – служебное поле, позволяющее значительно ускорить построение списков и поиск нужных элементов. Данная строка состоит из ID всех родительских элементов начиная с корневого (id: 0), заключенных в символы «/», а также уровня элемента после символа «#». Строка имеет следующий вид: /0/ID 1-го уровня/ID 2-го уровня/.../ IDN-го уровня/#N+1 (например, «/0/3/17/56/#4»). Для позиций указание уровня не обязательно.

Примеры

Пример 1

Получение дерева каталога из таблиц с подсчетом дочерних элементов и всех входящих позиций

```
<?

$tree = new TREE();

// разрешаем подсчет дочерних элементов
$tree->count_children = true;

// определяем таблицу категорий
$tree->categories_table = 'products_categories';

// определяем таблицу позиций
$tree->items_table = 'products';

// разрешаем подсчет особых позиций
$tree->count_special_items = true;

// поле, определяющее особые позиции
$tree->special_items_field = 'new';

// значение поля, определяющее особую позицию
$tree->special_items_value = '1';

// назначаем подсчет позиций по всей ветке
$tree->all_items = true;

Получаем и обрабатываем элементы
if( !$tree->CreateElementsFromTable( 0 ) ) { echo 'ошибка получения
элементов';die(); }
$tree->AddItemsCount();
```

```
// собираем объект
$tree->ConstructObject();

// печатаем дерево
PrintTree($tree,0);

?>
```

Пример 2

Получение дерева каталога из набора строк

```
<?

$categories = Array(
    0 => 'Компьютеры/Ноутбуки',
    1 => 'Бытовая техника/СВЧ/Аэрогриль',
    2 => 'Компьютеры/Ноутбуки/HP',
    3 => 'Компьютеры/Ноутбуки/Asus',
    4 => 'Бытовая техника/СВЧ/Аэрогриль',
    5 => 'Бытовая техника/СВЧ/Bosch'
);

$tree = newTREE();
$tree->CreateEmptyTree();
foreach( $categories as $str ) {
    $tree->AddLine($str);
}
$tree->ConstructObject();
PrintTree($tree,0);

?>
```

Результат:

```
[0] - Root
    [1] - Компьютеры
        [2] - Ноутбуки
            [6] - HP
            [7] - Asus
    [3] - Бытовая техника
        [4] - СВЧ
            [5] - Аэрогриль
            [8] - Bosch
```

Пример 3

Функция вывода дерева

```
<?

Function PrintTree($node,$int) {
    $space = $int*20;
    $sint = $int+1;
    echo '<span style="margin-left:'. $otstup.'px">['. $node->id.'] '.$node-
>name.' - ['.
$node->children_num.'] (['. $node->items_num.'['. $node-
>special_items_num.'])</span><br />';
    if (count($node->children)) {
        foreach ($node->children as $node) {
            PrintTree($node,$sint);
        }
    }
    return true;
}

?>
```

Сервисные классы (файл services.classes.php)

Класс TIMER

Данный класс предназначен для замера времени исполнения скриптов и их частей. Замер производится путем постановки начальной и конечной временных меток, и получения разницы между ними.

Методы класса

| |
|--|
| bool SetStart() |
| Устанавливает начальную точку отсчета. |
| bool SetStop() |
| Устанавливает конечную точку отсчета. |
| str GetResult() |
| Возвращает результат замера. |
| str GetCurrentValue() |
| Получение текущего промежуточного результата |

Класс LOGER

Данный класс предназначен для создания логов

Параметры объекта класса

| Параметр | Описание | По умолчанию |
|------------------|---------------------------|------------------------|
| str \$log_folder | Полный путь к файлу лога. | BASE_PATH.'/temp/logs' |

Методы класса

| | | |
|----------------------------------|------------------------|--|
| bool Open (\$file) | | |
| Создание или открытие файла лога | | |
| str \$file | Имя создаваемого файла | |

| bool PutLine(str \$string) | | |
|-----------------------------------|------------------|--|
| Вносит строку в файл лога. | | |
| str \$string | Текстовая строка | |
| bool Close() | | |
| Закрывает файл лога. | | |

Пример

```
<?
// создаем объекты
$log = new LOGGER();
$timer = new TIMER();

// запускаем таймер
$timer->Start();

// определяем директорию лога и открываем его
$log->log_folder = BASE_PATH.'logs/test/';
$log->Open('log.txt');

// пишем строку в лог
$log->PutLine('Начало работы');

/*
Выполнение каких-либо операций

*/

// заканчиваем логирование
$log->PutLine('Работа закончена');
$log->Close();

// останавливаем таймер и выводим результат
$timer->Stop();
echo $timer->GetResult();

?>
```

Статические функции (файл `functions.classes.php`)

В данном файле находятся статические функции для преобразования, валидации данных, а также другие служебные функции. Файл содержит только один класс.

Класс FUNC

Методы класса

| | | |
|---|---|--------|
| static str GrabRequestVar(\$method, \$var[, \$key[, \$html]]) | | |
| Возвращает обработанное значение переменной из POST или GET запроса. | | |
| str \$method | Метод запроса ("post" или "get") | |
| str \$var | Имя получаемой переменной или массива | |
| str \$key | При получении элемента массива в этой переменной указывается его ключ. | "" |
| bool \$html | Определяет является ли получаемое значение html кодом. Если установлено значение false все символы будут преобразованы в html сущности | false |
| static int GetItemID([\$position]) | | |
| Получает из адреса страницы и проверяет целочисленные ID элементов. | | |
| \$position | Позиция ID в адресной строке | 2 |
| static int GetNextID(\$table[, \$identifier[, \$type[, \$start]]) | | |
| Функция получения из таблицы следующего целочисленного ID для нового сохраняемого элемента. | | |
| str \$table | Таблица хранения элементов (без префикса) | |
| str \$identifier | Название поля хранения ID элементов | "id" |
| str \$type | Тип поиска следующего значения. Возможные значения: "next" - возвращается число на единицу большее, чем максимальное существующее; "empty" - поиск наименьшего незанятого значения ID начиная с числа, указанного в параметре start | "next" |

| | | |
|---|--|------|
| int \$start | Значение ID с которого начинается поиск пропущенных значений при типе поиска "empty" | 1 |
| static str SqlToUnixTime(\$timestamp) | | |
| Возвращает время в формате UNIX. | | |
| str \$timestamp | Дата и время в формате "yyyy-mm-dd hh:mm:ss" | |
| static str SqlToRusDate(\$date) | | |
| Возвращает дату в формате "dd.mm.yyyy". | | |
| str \$date | Дата в формате "yyyy-mm-dd" | |
| static str RusToSqlDate(\$date) | | |
| Возвращает дату в формате "yyyy-mm-dd". | | |
| str \$date | Дата в формате "dd.mm.yyyy" | |
| static str TimestampToRus(\$time[, \$ seconds]) | | |
| Возвращает дату в формате "dd.mm.yyyy hh:mm:ss". | | |
| str \$time | Дата в формате "yyyy-mm-dd hh:mm:ss" | |
| bool \$seconds | Включать ли секунды в результат | true |
| static str RusToTimestamp(\$time[, \$ seconds]) | | |
| Возвращает дату в формате "yyyy-mm-dd hh:mm:ss". | | |
| str \$time | Дата в формате "dd.mm.yyyy hh:mm:ss" | |
| bool \$seconds | Включать ли секунды в результат | true |
| static str ExtractTime(\$string[, \$seconds[, \$check]]) | | |
| Извлекает из строки «дата время» значение времени с возможностью проверки его на соответствие формату "hh:mm:ss" или "hh:mm". | | |
| str \$string | Строка «дата время» | |
| bool \$seconds | Включать ли секунды в результат | true |
| bool \$check | Проверять значение на соответствие | true |
| static str ExtractDate(\$string[, \$sql_date[, \$check]]) | | |
| Извлекает из строки «дата время» значение даты с возможностью проверки его на соответствие формату "dd.mm.yyyy" или "yyyy-mm-dd". | | |

| | | |
|--|---|-------|
| str \$string | Строка «дата время» | |
| bool \$sql_date | Дата в виде "yyyy-mm-dd" если true и в виде "dd.mm.yyyy" если false | false |
| bool \$check | Проверять значение на соответствие | true |
| static bool ClearDir(\$path) | | |
| Удаляет все содержимое директории. | | |
| str \$path | Полный путь к директории | |
| static bool RemoveDir(\$path) | | |
| Удаляет директорию и все ее содержимое. | | |
| str \$path | Полный путь к директории | |
| static bool CopyDir(\$path_from, \$path_to) | | |
| Копирует все содержимое директории. | | |
| str \$path_from | Полный путь к директории нахождения информации | |
| str \$path_to | Полный путь к директории назначения | |
| static bool IsValidEmail(\$string) | | |
| Проверяет, является ли строка e-mail адресом. | | |
| str \$string | Проверяемая строка | |
| static bool IsValidFileName(\$string) | | |
| Проверяет, является ли строка разрешенным в данной системе именем файла (должно состоять из латинских букв, цифр и знаков «-», «_»). | | |
| str \$string | Проверяемая строка | |
| static bool IsValidPictureName(\$string) | | |
| Проверяет, является ли строка разрешенным в данной системе именем файла изображения. | | |
| str \$string | Проверяемая строка | |
| static bool IsInteger(\$string) | | |
| Проверяет, является ли строка целочисленным значением. | | |
| str \$string | Проверяемая строка | |
| static bool IsFloat(\$string) | | |

| | | |
|--|--|------|
| Проверяет, является ли строка числом с плавающей точкой. | | |
| str \$string | Проверяемая строка | |
| static bool IsSqlDate(\$date) | | |
| Проверяет строку на соответствие формату "yyyy-mm-dd". | | |
| str \$date | Проверяемая строка | |
| static bool IsDate(\$date) | | |
| Проверяет строку на соответствие формату "dd.mm.yyyy". | | |
| str \$date | Проверяемая строка | |
| static bool IsTime(\$time[, \$seconds]) | | |
| Проверяет строку на соответствие формату "hh:mm:ss" или "hh:mm". | | |
| str \$time | Время в формате "hh:mm:ss" | |
| bool \$seconds | Включены ли секунды | true |
| static str TextAreaToHtml(\$string) | | |
| Преобразует строковое значение, полученное из элемента формы textarea для его хранения в базе и отображения в браузере с сохранением переносов строк. | | |
| str \$string | Преобразуемая строка | |
| static str HtmlToTextArea(\$string) | | |
| Преобразует строковое значение, полученное из базы данных для отображения в элементе формы textarea с сохранением переносов строк. | | |
| str \$string | Преобразуемая строка | |
| static array GetWrongSearchVariants(\$string) | | |
| Функция, используемая для организации поиска по тексту с учетом возможных орфографических ошибок. Возвращает массив вариантов возможных ошибок в строке (пропуск буквы, лишняя буква, неверная буква), элементы которого могут быть использованы при sql выборке оператором LIKE | | |
| str \$string | Преобразуемая строка | |
| static str PriceSpacer(\$s[, \$spacer]) | | |
| Вставляет разделители между тысячными разрядами числа (например, для вывода цен). | | |
| str \$string | Преобразуемая строка, представляющая собой целое или дробное число | |

| | | |
|--|------------------------------|--------------|
| str \$spacer | Разделитель | ' ' (Пробел) |
| static array HexToRgb(\$color)* | | |
| Преобразует цвет из HEX представления в RGB. Возвращаемый результат содержит массив из трех элементов содержащих определения цветов. Ключи массива 'red', 'green', 'blue'. | | |
| str \$color | Цвет в HEX | |
| static str RgbToHex(\$r, \$g, \$b)* | | |
| Преобразует цвет из RGB представления в HEX. Возвращаемый результат содержит строку с определением цвета (например, "#acf238"). | | |
| int \$r | Красная составляющая (0-255) | |
| int \$g | Зеленая составляющая (0-255) | |
| int \$b | Синяя составляющая (0-255) | |

* - Доступны с версии файла 1.1.

Стандартные SQL таблицы

IPLF дает возможность использовать в объектах методы для работы со стандартными SQL таблицами. Они позволяют, например, построить более структурированную систему компонентов, работу с SEF адресами, хранить настройки, стандартизировать хранение базы пользователей. Использование этих таблиц не обязательно, но делает работу с FrameWork более оптимизированной. Использование таблиц устанавливается в файле config.php.

При использовании стандартных таблиц обязательными являются поля, описанные в спецификациях ниже. Однако к таблице могут быть добавлены любые другие поля.

Для быстрого создания необходимых таблиц вы можете использовать файл `_st_tables_create.php`, находящийся в директории FrameWork. При этом необходимо изначально проверить и при необходимости изменить директории включаемых файлов в соответствии с конфигурацией Вашего сайта.

Спецификация таблиц

COMPONENTS

Таблица перечня компонентов

| Название | Тип | Описание |
|-------------|--------------|--|
| id | Int(3) | ID компонента в системе |
| component | Varchar(255) | Универсальный идентификатор компонента. Совпадает с директорией расположения файлов компонента на диске. |
| name | Varchar(255) | Название компонента (в произвольной форме) |
| description | Text | Описание компонента |
| available | Tinyint(1) | Доступность компонента |
| version | Int(3) | Версия компонента |
| edition | Int(3) | Редакция текущей версии компонента |
| license | Varchar(255) | Код лицензии при ее использовании |
| last_update | Timestamp | Дата и время последнего обновления |

SECTIONS

Таблица перечня разделов

| Название | Тип | Описание |
|-----------|--------------|--|
| id | Int(3) | ID раздела в системе |
| section | Varchar(255) | Идентификатор раздела. Должен быть определен в контроллере компонента. |
| name | Varchar(255) | Название раздела (в произвольной форме) |
| component | Int(3) | Описание раздела |
| area | Int(3) | Доступность раздела (зона доступа) |
| available | Tinyint(1) | Доступность раздела |

SETTINGS

Таблица установок

| Название | Тип | Описание |
|----------|--------------|-------------------|
| name | Varchar(255) | Название свойства |
| value | Varchar(255) | Значение свойства |

ALIASES

Таблица псевдонимов ЧПУ ссылок

| Название | Тип | Описание |
|----------|--------------|---------------|
| alias | Varchar(255) | Псевдоним |
| url | Varchar(255) | Чистая ссылка |

USERS

Таблица пользователей

| Название | Тип | Описание |
|-------------------|--------------|--|
| id | Int(10) | ID пользователя |
| login | Varchar(255) | Логин для входа в систему |
| name | Varchar(255) | Имя пользователя |
| pass | Varchar(255) | Пароль пользователя (название поля может быть изменено если изменить значение соответствующей константы в файле config.php) |
| key | Varchar(255) | Ключ безопасности пользователя (название поля может быть изменено если изменить значение соответствующей константы в файле config.php) |
| group | Int(10) | Группа пользователя |
| email | Varchar(255) | E-mail пользователя |
| registration_date | Timestamp | Дата и время регистрации |
| approved | Tinyint(1) | Пользователь одобрен |
| active | Tinyint(1) | Пользователь активен |

USERS_GROUPS

Описание групп пользователей

| Название | Тип | Описание |
|-------------|--------------|--|
| id | Int(3) | ID группы |
| name | Varchar(255) | Имя группы |
| identifier | Varchar(255) | Идентификатор группы (латинскими буквами) |
| description | Text | Описание |
| areas | Varchar(255) | ID доступных областей доступа через слэш (например, "/1/3/") |
| active | Tinyint(1) | Группа активна |

USERS_GROUPS_ACCESS

Уровни доступа групп пользователей к разделам

| Название | Тип | Описание |
|----------|--------------|--|
| group | Int(3) | ID группы |
| area | Int(3) | ID раздела |
| section | Varchar(255) | Идентификатор раздела. Должен быть определен в контроллере компонента. |
| access | Int(1) | Цифровое определение типа доступа |

SESSIONS

Список сессий при авторизации через cookie

| Название | Тип | Описание |
|----------|--------------|---------------------------------------|
| sid | Varchar(255) | ID сессии |
| user | Int(20) | ID пользователя |
| data | Text | Сериализованный массив \$_SESSION |
| started | Int(20) | Время начала сессии в формате UNIX |
| expire | Int(20) | Время истечения сессии в формате UNIX |

Полный список констант и базовых параметров

Задаваемые константы

SCRIPT_FOLDER - Директория установки скриптов относительно корневой домена. Если это корневая директория, то указывается "/", если внутренняя, то "/folder/" или "/folder/subfolder/".

WEB_PROTOCOL - web протокол, например, "http://" или "https://"

REF_END - Окончание ссылки (например, "/", ".html")

Настройки базы данных:

DB_HOST - хост базы

DB_USER - пользователь

DB_PASS - пароль пользователя

DB_BASE - имя базы данных

DB_PREFIX - префикс таблиц

SECURITY_KEY - Ключ безопасности сайта. Используется для шифрования конфиденциальных данных.

Кодировки:

DOCUMENT_CHARSET - в виде "windows-1251", "utf-8" и т.д.

DB_CHARSET - в виде "cp1251", "utf8" и т.д.

COLLATION_CHARSET – в виде "cp1251_general_ci", "utf8-unicode_ci" ит.д.

WFCONFIG_ARCHITECTURE –тип архитектуры IPLF

WFCONFIG_COMPONENTS_FOLDER – директория расположения компонентов

WFCONFIG_VIEW_FOLDER - директория расположения шаблонов

WFCONFIG_COMPONENTS_URI - путь url директории компонентов (от корневой директории сайта без последнего /, например, 'components')

WFCONFIG_VIEW_URI - путь url директории шаблонов (от корневой директории сайта без последнего /, например, 'view')

Константы использования стандартных SQL таблиц IPLF

WFCONFIG_USE_ALIASES_TABLE - использовать таблицу псевдонимов ссылок.

WFCONFIG_GET_SETTINGS_TABLE - считывать таблицу установок

WFCONFIG_USE_SECTIONS_TABLE - использовать для построения кэша компонентов стандартную таблицу

Регистрация и пользователи

WFCONFIG_INITIATE_REGISTRED_USER - проверять вошедшего пользователя на регистрацию в соответствии со стандартной таблицей пользователей

WFCONFIG_USE_COOKIE_AUTHORIZATION – использовать авторизацию через cookie. Используется стандартная таблица сессий**.

WFCONFIG_SESSION_TIME – время жизни сессии при использовании авторизации через cookie

WFCONFIG_USER_AVATAR_FOLDER - директория хранения аватаров

WFCONFIG_USER_AVATAR_EXTENSION – расширение аватаров

WFCONFIG_USER_PASS_FIELD - название поля пароля пользователя в таблице пользователей ***

WFCONFIG_USER_KEY_FIELD - название поля ключа пользователя в таблице пользователей

Формируемые константы

Константы, определяемые при инициации ядра.

BASE_URL – адрес основной страницы сайта, включающий директорию установки скриптов (например, «<http://site.ru/folder/>»или «<https://site.ru/>»)

BASE_PATH – полный путь к директории установки (например, «home/owner/data/www/site.net/»)

COMPONENTS_URL – адрес расположения компонентов

COMPONENTS_PATH – полный путь к директории компонентов

VIEW_URL – адрес расположения шаблонов для текущей области доступа

VIEW_PATH – полный путь к директории расположения шаблонов для текущей области доступа

Базовые параметры

Ниже перечислены элементы массива, находящегося в параметре \$parameters класса FRAME_COREи формируемые при его инициации в любом случае.

\$show_errors – передает настройку отображения ошибок в PHP

\$access_areas_array – содержит массив ID областей доступа

\$url_variables_real – содержит массив строк, содержащихся в адресе страницы через «/»

\$url_variables – содержит массив строк, адреса страницы после преобразования псевдонима в системный адрес страницы. Если псевдоним не используется, равен значениям \$url_variables_real

\$pagination_position – номер текущей страницы при постраничном выводе. Номер страницы определяется по параметру /p=n/, автоматически добавляемому в конец адреса страницы, где n – номер страницы

\$authorized – авторизован ли текущий пользователь (false/true)

Журнал изменений

Rev 1

Начальная редакция.

Rev 2

1. В файл `functions.classes.php` добавлены методы
 - `FUNC::HexToRgb`
 - `FUNC::RgbToHex`.
2. Метод `FRAME_CORE::Error404` стал статическим

Rev 3

1. В файл `base.classes.php` добавлен метод `FRAME_CORE::UV()`
2. В файл `images.classes.php` добавлены методы
 - `IMAGE::AntiAliasing()`
 - `IMAGE::Point()`
 - `IMAGE::Fill()`
 - `IMAGE::Pixel()`
 - `IMAGE::Line()`
 - `IMAGE::Rectangle()`
 - `IMAGE::Polygon()`
 - `IMAGE::Ellipse()`
 - `IMAGE::Arc()`
 - `IMAGE::Sector()`
 - `IMAGE::Text()`
 - `IMAGE::FillContour()`