



Руководство программиста  
API – общее описание.

[www.amiro.ru](http://www.amiro.ru)

Введение .....	4
Общая схема объектов API и точки входа Amiro.CMS .....	5
3.1 Режимы работы отдельно стоящего скрипта .....	7
Простые примеры работы с новостями.....	8
4.1 Получение новости.....	8
4.2 Получение списка новостей .....	8
4.3 Рабочий пример вывода списка новостей.....	9
4.3.1 Исходный код.....	9
4.3.3 Список доступных полей модели .....	11
4.3.4 Объект AMI_Response .....	12
4.3.5 Шаблонизатор AMI_Template, AMI_TemplateSystem.....	12
4.3.6 Комментарии .....	14
MVC API.....	15
Пример взаимодействия объектов.....	15
Модели (Model).....	16
7.1 Модель данных (Table) .....	16
7.2 Модель списка (TableList) .....	17
7.3 Модель элемента (TableItem).....	17
7.4 Зависимость (соединение) моделей (JOIN) .....	18
7.5 Вычисляемые и виртуальные поля .....	19
7.6 Валидаторы.....	20
Ресурсы.....	22
8.1 Существующие ресурсы.....	22
Компоненты.....	22
Общая информация .....	22
Табличный список .....	26

Создание списка .....	26
Определение списка и порядка полей.....	27
Форматтеры.....	29
Действия .....	29
Групповые действия .....	29
Примеры реализации.....	30
Форма.....	34
Создание формы .....	34
Определение списка и порядка полей.....	35
Валидация полей формы на стороне клиента.....	36
Вкладки.....	37
Примеры реализации.....	38
Фильтр .....	39
Создание фильтра.....	39
Определение списка и порядка полей.....	40
Примеры реализации.....	42
Статусные сообщения .....	43
Пример реализации модуля (MVC подход).....	46
События .....	68
10.1 События получения списка элементов.....	68
10.1.1 on_table_get_list.....	68
10.1.2 on_query_add_table .....	69
10.1.3 on_query_add_joined_columns.....	70
10.1.4 on_list_recordset.....	71
10.1.5 on_before_set_data_model_item.....	72
10.2 on_html_meta_change .....	72

Средства отладки и профилирования.....	73
11.1 Включение отладки .....	73
11.2 Профилирование кода и запросов к БД.....	74
11.3 Средства отладки шаблонов.....	75

## Введение

Начиная с версии Amiro.CMS 5.10 доступно API разработчика. [Beta] версия означает, что при дальнейших обновлениях потенциально возможны какие-то изменения, требующих незначительной модификации разработанного функционала. Вероятность таких ситуаций сводится к минимуму.

Данное руководство построено по нарастанию уровня сложности используемых методов и схем работы.

В главе «Точки входа» представлена информация начального уровня: общая схема взаимодействия объектов Amiro.CMS, описаны варианты включения пользовательского кода, рассмотрены примеры инициализации API.

В главе «Простые примеры работы с новостями» приведены несколько практических примеров, раскрывающих базовые возможности использования API, такие, как работа с моделями и управление выводом.

На этом этапе Вы сможете создавать простые скрипты получения и отображения списка элементов (новостей, статей, товаров и т.п.).

Глава «MVC API» познакомит с возможностями использования расширяемой MVC-архитектуры для реализации решений со сложной логикой.

Раздел «Пример взаимодействия объектов» на схематичном примере реализации плагина PlgAJAXResponder даст первичное представление об устройстве типового решения построенного на MVC архитектуре.

В разделе «Модели» будут подробно рассмотрены используемые типы моделей, организация связей между различными моделями, использование настраиваемых и вычисляемых полей модели, а так же валидация данных.

Раздел «Ресурсы» описывает основные возможности работы с ресурсами, создание ресурсов, получение ресурса по имени.

Раздел «Компоненты» познакомит с основными компонентами Amiro.CMS: списками, формами и фильтрами. В этой главе будут описаны возможности связанные с

формированием и управлением компонентами, приведены практические примеры реализации.

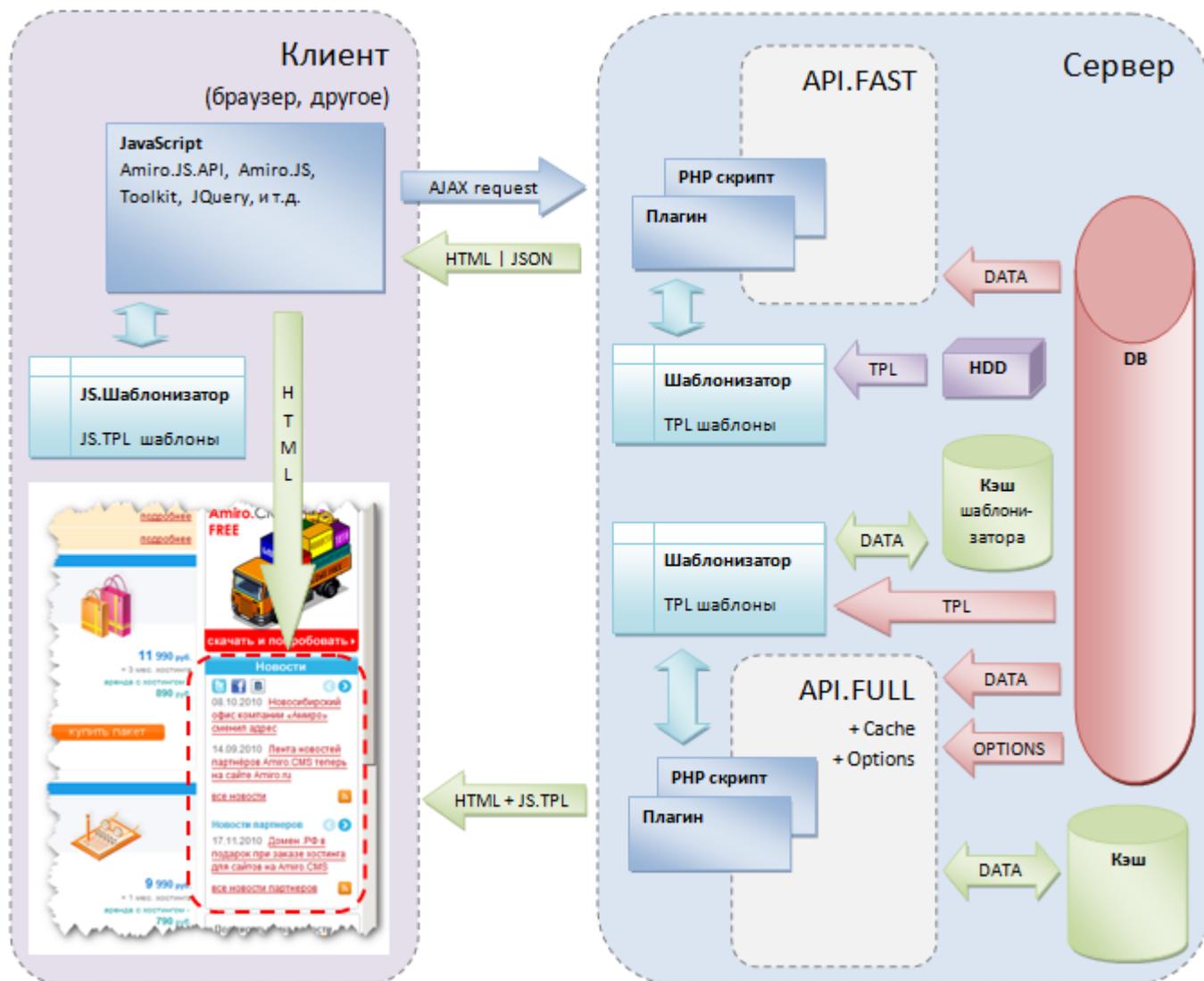
В разделе «Пример реализации плагина 6.0» приведен пример реализации небольшого плагина, использующего большую часть из описанных в предыдущих главах возможностей API.

В разделе «События» приведено описание и практические примеры использования системы событий, позволяющей максимально использовать возможности API для управления данными.

Глава «Средства отладки и профилирования» познакомит с соответствующим набором инструментов и приемами для удобной работы.

## Общая схема объектов API и точки входа Amiro.CMS

Диаграмма взаимодействия объектов Amiro.CMS



На данный момент предлагаются 2 основные точки входа для размещения собственного кода:

1. Отдельно стоящий PHP скрипт
2. Плагин (plugin) для Amiro.CMS

*В последующих версиях API будут добавлены новые точки входа, предоставляющие разработчикам еще более широкие возможности по интеграции собственного кода в систему.*

**Отдельно стоящий PHP скрипт** — это файл, содержащий пользовательский PHP код, использующий возможности API Amiro.CMS. Данную точку входа можно использовать для получения данных браузером в виде HTML, JSON или прочем формате посредством Amiro.JS.API или другой AJAX-библиотеки; для создания скриптов, выполняемых в

фоновом режиме (например, генерация отчетов, обмен данными и т.п.). В текущей версии API скрипту недоступно кеширование, по этой причине не следует нагружать скрипты тяжелым функционалом с частыми вызовами, т.к. это вызовет неоптимальную загрузку сервера. Amiro.CMS ведет учет времени выполнения скрипта и при превышении им заданной величины – в системный лог записывается предупреждение. Различные способы, типы использования и способы создания своего кода описаны в [API.Reference/Environment setting up entry point](#) [English]

**Плагин (Plugin) для Amiro.CMS** — это набор из одного или нескольких модулей, спецблоков для Amiro.CMS, который может быть представлен в виде дистрибутива для автоматической установки в систему. Плагины могут распространяться в качестве самостоятельных дистрибутивов и с мастером установки через панель управления. Плагины автоматически кешируются системой.

Создание плагина подробно описано в соответствующем руководстве: [Разработка и установка плагинов](#)

Независимо от точки входа, доступны для использования одни и те же методы и ресурсы. Т.е. классы, разработанные для плагина, могут быть использованы в отдельно стоящем скрипте, и наоборот.

При разработке, следует помнить, что результаты работы как отдельно стоящего скрипта, так и плагина могут кешироваться внутренним кешем Amiro.CMS. Механизмы отладки шаблонов и отключения кеширования рассмотрены в главе «[Средства Отладки Amiro.CMS](#)»

### 3.1 Режимы работы отдельно стоящего скрипта

Для запуска отдельно стоящего скрипта, необходимо подключить файл `ami_env.php`, находящийся в корне сайта. Для управления режимом работы можно указать параметры подключения в массиве `$AMI_ENV_SETTINGS`.

Допустимые значения массива и их расшифровка:

Параметр	Значение	Значение по умолчанию	Описание
<code>response_type</code>	'HTML' 'JSON'	HTML	Формат выдачи данных (для формирования заголовков)
<code>response_buffered</code>	true false	true	Если необходимо отключить буферизацию вывода, можно отключить этот параметр.

Пример:

```
$AMI_ENV_SETTINGS = array(  
    "response_type" => "JSON",  
    "response_buffered" => "false",  
);  
require 'ami_env.php';  
...
```

Дополнительные примеры способов запуска и инициализации отдельно стоящих скриптов рассмотрены в [API.Reference/ Environment setting up entry point](#) [English]

## Простые примеры работы с новостями

В этой главе будут рассмотрены несколько примеров работы с новостями (получение новости, получение списка новостей) и классы для генерации результатов вывода.

В приведенных ниже примерах показана только та часть кода, которая непосредственно отвечает за получение требуемых данных. Точка входа (отдельно стоящий скрипт или плагин) может быть любой.

### 4.1 Получение новости

Получение новости с идентификатором id=1 и вывод данных:

*Используемые методы будут пояснены в дальнейших примерах.*

```
$oNewItem = AMI::getResourceModel('news/table')->find(1);  
echo "News item ID = ".$oNewItem->id."<br>";  
print_r(iterator_to_array($oNewItem));
```

### 4.2 Получение списка новостей

Получение значений столбцов id, header у трех новостей и печать данных.

```
$oNewsList = AMI::getResourceModel('news/table')->getList()  
->addColumn(array('id', 'header'))  
->setLimitParameters(0, 3)->load();  
print_r(array_map('iterator_to_array', iterator_to_array($oNewsList)));
```

Или поэлементная печать, используя foreach:

```
$oNewsList = AMI::getResourceModel('news/table')
->getList()->addColumns(array('id', 'header'))
->setLimitParameters(0, 3)->load();
foreach($oNewsList as $oNewsItem) {
    print_r(iterator_to_array($oNewsItem));
}
```

### 4.3 Рабочий пример вывода списка новостей

Предполагается разработка на домене cms.my.

Задача – создать скрипт my\_news.php, выводящий 3 опубликованных русскоязычных новости в виде таблицы, состоящей из 2 столбцов: дата, название. Т.е.

[http://cms.my/my\\_news.php](http://cms.my/my_news.php) выводит требуемую таблицу новостей.

#### 4.3.1 Исходный код

Поскольку файл содержит код HTML, необходимо сохранять его в кодировке UTF-8.

```
<?php
require 'ami_env.php';

// Get response object
$oResponse = AMI::getSingleton('response');
$oResponse->start();

/*
```

Код, находящийся между комментариями «Show items list start» и «Show items list end» отвечает непосредственно за генерацию списка элементов.

```
*/
// *****
// Show items list start
// *****

$modId = 'news';

/*
```

Получение объекта модели для списка:

```
*/
$oNewsModelList = AMI::getResourceModel($modId . '/table')->getList();

/*
```

На список накладываются фильтры, порядок сортировки, лимиты, выбираются поля объектов, которые должны быть загружены, и происходит загрузка элементов из базы данных:

```
*/  
  
$oNewsModelList  
->addColumn(array('date_created', 'header', 'id'))  
->addOrder('date_created', ' desc')  
->addWhereDef('AND ' . $oNewsModelList->getFieldName('public') . ' = 1')  
->addWhereDef(  
    DB_Query::getSnippet('AND %s = %s')  
    ->plain($oNewsModelList->getFieldName('lang'))  
    ->q('ru')  
)  
->setLimitParameters(0, 3)  
->load();
```

```
/*
```

Производится подготовка переменной, для сбора таблицы результатов вывода:

```
*/  
  
$res = '<table>';  
$res .= '<tr><th>Дата</th><th>Название</th></tr>';
```

```
/*
```

Цикл по списку элементов:

Следует обратить внимание, что `$oNewsModelItem`, получаемый в цикле, это объект модели элемента.

```
*/  
  
foreach($oNewsModelList as $oNewsModelItem) {  
    $res .= '<tr><td>' . $oNewsModelItem->date_created . '</td><td>' .  
    $oNewsModelItem->header . '</td></tr>';  
}
```

```
/*
```

Закрытие таблицы:

```
*/  
  
$res .= '</table>';  
  
// *****  
// Show items list end  
// *****  
$oResponse->write($res);
```

```
$oResponse->send();
```

#### 4.3.3 Список доступных полей модели

Список доступных полей модели возвращает метод

[AMI ModTable::getAvailableFields\(\)](#):

```
AMI::getResourceModel($modId . '/table')->getAvailableFields();
```

Общие поля, присутствующие в большинстве моделей:

Поле	Тип	Описание
Общие поля		
id	целое	Идентификатор
public	0/1	1 – элемент доступен, 0 – элемент неопубликован
archive	0/1	Статус архивности
header	строка	Заголовок
announce	строка	Краткое описание
body	строка	Полное описание
lang	строка, 2-3 символа	Принадлежность элемента к локализации (ru, en, de, и т.д.)
date_created	дата	Дата создания
date_modified	дата	Дата модификации
sublink	строка	Собственная часть URL элемента на публичной части сайта
id_page	Целое	Идентификатор страницы элемента в менеджере сайте (0 – общий элемент)
id_owner	Целое	Идентификатор владельца элемента
position	Целое	Позиция для сортировки в ручном режиме
details_noindex	0/1	Флаг, регулирующий запрет индексации поисковиками деталей
hide_in_list	0/1	Флаг, регулирующий скрытие элемента в списках, без запрета на показ по прямой ссылке.
sticky	0/1	Флаг, указывающий, что элемент является прикрепленным в списке
date_sticky_till	строка, дата	Срок действия флага прикрепленности
Поля расширения изображений		

ext_img	строка	Файл изображения
ext_img_small	строка	Файл малого изображения
ext_img_popup	строка	Файл всплывающего изображения
Поля расширения рейтингов		
ext_rate_count	целое	Число голосов
ext_rate_rate	целое	Рейтинг элемента
Поля расширения обсуждения		
ext_dsc_disable	0/1	Флаг, регулирующий запрет обсуждения элемента
Поля расширения категорий (добавляются только при активированном расширении категорий)		
cat_id	целое	Идентификатор категории
cat_header	строка	Заголовок категории
cat_sublink	строка	Собственная часть URL категории элемента на публичной части сайта

Данный метод может использоваться для проверки наличия полей у требуемой модели.

#### 4.3.4 Объект AMI\_Response

Объект [AMI\\_Response](#) предназначен для управления выводом: вывод HTTP-заголовков, буферизация, кеширование, замеры и отладка скорости работы.

Объект создается автоматически при работе всех стандартных модулей Amiro.CMS, и его настоятельно рекомендуется использовать при создании собственных скриптов.

Создание объекта:

```
// Get response object
$oResponse = AMI::getSingleton('response');
$oResponse->start();
```

Вывод результата в требуемом формате (JSON или HTML):

```
$oResponse->write($res);
$oResponse->send();
```

#### 4.3.5 Шаблонизатор AMI\_Template, AMI\_TemplateSystem

В приведенном выше примере для упрощения кода не используются шаблоны и логика смешана с отображением. В реальном коде делать так крайне не рекомендуется. API предоставляет два вида шаблонизаторов – полный и облегченный. Пример использования будет показан в MVC разделе данного руководства.

При разработке для публичной части сайта рекомендуется использовать некеширующий шаблонизатор [AMI Template](#), работающий только с шаблонами на диске, содержащий минимально достаточный функционал, потребляющий минимум памяти и процессорного времени. Полный шаблонизатор [AMI TemplateSystem](#) рекомендуется использовать, только если базовым функционалом обойтись заведомо невозможно..

В плагине достаточно использовать AMI\_Template в случаях если:

- Плагин является некешируемым, важна скорость обработки шаблона (типовой пример – плагин для отдачи быстрых AJAX запросов).
- Шаблоны содержат только заменяемые переменные типа , и т.д. Поддержка специальных конструкций , , и т.п. в лёгком шаблонизаторе отсутствует.
- Шаблон не содержит конструкции подключения языковых файлов, например `%%include_language...%%`. Все языковые файлы должны быть добавлены в PHP коде плагина. Тем не менее языковые переменные типа `%%caption%%` обрабатываются.
- Шаблоны лежат только на жёстком диске.

#### Типовые операции с шаблонами:

Операция	Пример вызова
Создание объекта	<code>\$oTpl = new AMI_Template;</code>
Добавление блока	<code>oTpl-&gt;addBlock('my_block', 'templates/my_tpl.tpl');</code>
Парсинг шаблона	<code>\$str = \$oTpl-&gt;parse('my_block:my_set', \$aScope);</code>
Добавление блока	<code>oTpl-&gt;addBlock('my_block', 'templates/my_tpl.tpl');</code>
Парсинг языкового файла	<code>\$aLang = \$oTpl-&gt;parseLocale('templates/lang/my_lang.lng');</code>

AMI\_TemplateSystem в отличие от AMI\_Template поддерживает чтение шаблонов из базы и позволяет использовать полный функционал системы.

#### Типовые операции с шаблонами:

Операция	Пример вызова
Создание объекта	<code>\$oTpl = new AMI_TemplateSystem;</code>
Переключение источника шаблонов	<code>\$oTpl-&gt;addBlock('templates/', 'db');</code>
Добавление блока	<code>\$oTpl-&gt;addBlock('my_block', 'templates/my_tpl.tpl');</code>
Парсинг шаблона	<code>\$str = \$oTpl-&gt;parse('my_block:my_set', \$aScope);</code>
Парсинг языкового файла	<code>\$aLang = \$oTpl-&gt;parseLocale('templates/lang/my_lang.lng');</code>

## 4.3.6 Комментарии

### 4.3.6.1 Перенос кода примера в плагин

Следует обратить внимание, что для того, чтобы создать плагин, выводящую аналогичную таблицу, достаточно код, размещенный между комментариями «Show items list start» и «Show items list end» добавить, например, в качестве спецблока плагина, и вернуть результат в переменную \$resultHtml.

Т.е. если в спецблоке плагина используется файл my\_speblock.php, то он должен выглядеть так:

```
<?php

// *****
// Show items list start
// *****

...

// *****
// Show items list end
// *****

$resultHtml = $res;
```

### 4.3.6.2 Автоматическая генерация изображений

Если при разработке скриптов с использованием точки входа ami\_env.php требуется функциональность автоматической генерации изображений, заложенная в системе, параметры автоматической генерации, соответствующие одноимённым настройкам модулей в секции «Расширение «Изображения»», необходимо задавать следующим образом:

```
<?php

$modId = 'news';

// Выбранные изображения будут уменьшены автоматически
AMI::setOption($modId, 'generate_pictures', array('picture', 'popup_picture',
'small_picture'));

// Приоритетное исходное изображение
AMI::setOption($modId, 'prior_source_picture', 'popup_picture');
```

```
// Максимальная ширина уменьшенного изображения
AMI::setOption($modId, 'picture_maxwidth', 300);

// Максимальная высота уменьшенного изображения
AMI::setOption($modId, 'picture_maxheight', 300);

// Максимальная ширина маленького изображения
AMI::setOption($modId, 'small_picture_maxwidth', 80);

// Максимальная высота маленького изображения
AMI::setOption($modId, 'small_picture_maxheight', 80);

// Максимальная ширина всплывающего изображения
AMI::setOption($modId, 'popup_picture_maxwidth', 800);

// Максимальная высота всплывающего изображения
AMI::setOption($modId, 'popup_picture_maxheight', 600);
// Увеличивать ли изображение, если оно меньше чем задано
AMI::setOption($modId, 'generate_bigger_image', true);
```

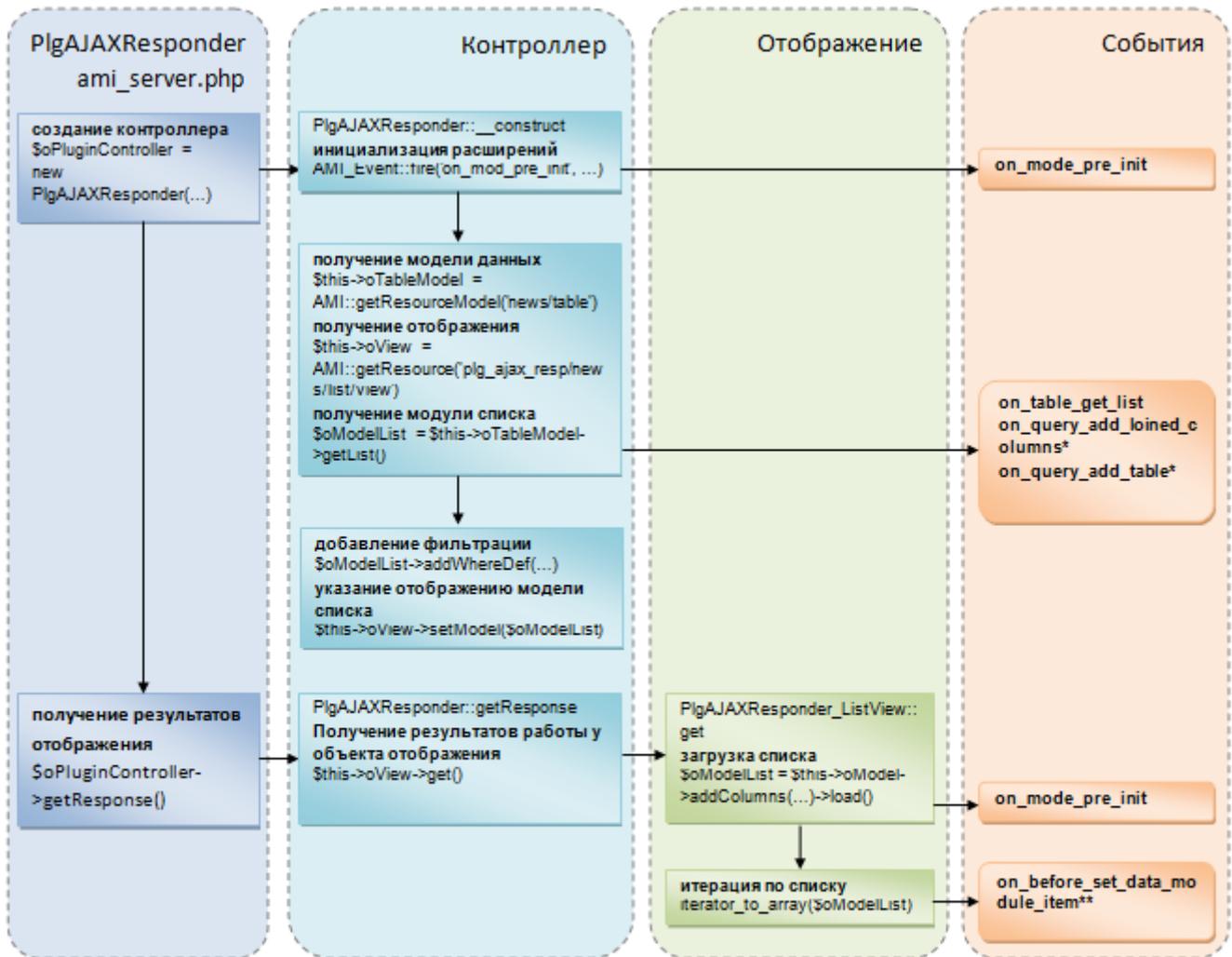
## MVC API

Для более полного использования API и написания расширенного функционала необходимо использовать API на MVC основе. (<http://ru.wikipedia.org/wiki/MVC>). В системе явным образом выделяются модель, контроллер и отображение. Все объекты описываются как *ресурсы*, создание и управление которыми происходит через централизованный класс управления ресурсами.

Взаимодействия могут быть двух видов: явное (прямой вызов) и неявное (обработка событий). Взаимодействия могут быть двух видов: явное (прямой вызов) и неявное (обработка событий).

## Пример взаимодействия объектов

На схеме отображено подробное взаимодействие объектов в плагине PlgAJAXResponder входящем в поставку Amiro.CMS, который может быть настроен на возвращение списка новостей в формате JSON.



\* - в случае наличия зависимых моделей, дополнительно также вызываются on\_query\_add\_joined\_columns и on\_query\_add\_table по 1 разу для каждой зависимой модели (для новостей зависимых моделей нет, событие не вызывается)  
 \*\* - событие on\_before\_set\_data\_model\_item вызывается столько раз, сколько записей возвращается моделью списка

Рисунок 1.

## Модели (Model)

### 7.1 Модель данных (Table)

Модель предоставляет разработчику интерфейс для доступа к данным. При этом, модель данных не хранит непосредственно данные, этим занимается модель элемента. Но модель данных умеет этими данными управлять. Например, выдавать данные в некотором внешнем формате, а сохранять в удобном внутреннем формате.

Модель данных выполняет следующие роли:

1. Создает модель списка

2. Создает модель элемента
3. Управляет зависимостью моделей (добавление, активация / деактивация)
4. Возвращает информацию о модели (столбцы таблицы базы данных, атрибуты файлов и т.п.)

Следует обратить внимание, что ресурсы моделей списка и элемента можно получить непосредственно из хранилища ресурсов, но верным является получение данных моделей через модель данных.

Пример получения модели данных для модуля новостей:

```
$oNewsTableModel = AMI::getResourceModel('news/table');
```

Результат: экземпляр объекта [News Table](#)

## 7.2 Модель списка (TableList)

Модель списка отвечает за получение списка элементов и предоставляет средства для его обхода.

Модель списка не хранит данные, этим занимается модель элемента.

Модель списка выполняет следующие роли:

1. Наложение фильтров на данные
2. Наложение лимитов на количество возвращаемых данных
3. Указание данных, которые необходимо получить

Модель списка используется для загрузки данных списка, который будет передан для отрисовки сущности отображения. Проще всего модель списка понимать, как итератор по данным, возвращающий модели элементов, хранящих данные, или выборку нескольких записей одной таблицы из базы данных.

Пример получения модели списка для модуля новостей:

```
$oNewsModelList = AMI::getResourceModel('news/table')->getList();
```

Результат: экземпляр объекта [News TableList](#)

## 7.3 Модель элемента (TableItem)

Модель элемента отвечает за хранение данных.

Модель списка выполняет следующие роли:

1. Загрузку данных в класс из хранилища
2. Сохранение данных из класса в хранилище
3. Удаление одного элемента из хранилища
4. Получение свойств(а) загруженного элемента
5. Установка свойств(а) загруженному элементу
6. Установка списка свойств, для которых будет запоминаться исходное состояние и получение изменившихся свойств

Проще всего модель элемента понимать, как одну запись одной таблицы в базе данных.

Пример получения модели нового элемента для модуля новостей:

```
$oNewsModelItem = AMI::getResourceModel('news/table')->getItem();
```

Результат: экземпляр объекта [News TableItem](#).

Пример задания неквотируемого значения поля:

```
class AmiSample_TableItemModifier extends AMI_ModTableItemModifier{
    // Добавляем неквотируемое поле `ts`
    public function getDefaultsOnSave($onCreate){
        $aDefaults = parent::getDefaultsOnSave($onCreate);
        $aDefaults['append']['ts'] = DB_Query::getSnippet('%s')-
>plain('NOW()');
        $aDefaults['overwrite']['ts'] = DB_Query::getSnippet('%s')-
>plain('NOW()');
        return $aDefaults;
    }
    // ...
}

// ....

$oTableItem = AMI::getResourceModel('ami_sample/table')->getItem();
// Устанавливаем значение
$oTableItem->ts = DB_Query::getSnippet('DATE_ADD(NOW(), INTERVAL 5 day');
```

## 7.4 Зависимость (соединение) моделей (JOIN)

Зависимость моделей — это возможность получения данных из другой модели, связанной с текущей по какому-либо правилу.

Зависимость бывает активной и неактивной. Активная зависимость указывает на то, что при получении списка элементов (одного элемента), необходимо произвести соединение моделей, и вернуть результаты, содержащие данные главной и зависимой моделей. Неактивная зависимость указывает на то, что существует потенциальная возможность

соединения моделей. Все зависимости не активны по умолчанию, их необходимо активизировать явным образом.

Проще всего активную зависимость моделей понимать, как соединение двух таблиц в базе данных, а неактивную, как возможность такого соединения.

Пример добавления зависимости моделей (статей и категорий статей) метод [setDependence\(\)](#)

```
class Articles_Table extends AMI_ModTable{
    public function __construct(){
        $this->setDependence('articles_cat', 'cat', 'cat.id=i.id_cat');
        ...
    }
    ...
}
```

Пример активации зависимости моделей статей и категорий статей (только после данной конструкции, будет происходить соединение таблиц) метод [setActiveDependence](#)

```
AMI::getResourceModel('articles/table')->setActiveDependence('cat');
```

## 7.5 Вычисляемые и виртуальные поля

Вычисляемые поля – это механизм преобразования данных из внутреннего формата хранения во внешний формат. Типовой пример вычисляемого поля – «дата создания», которая хранится в БД в mysql формате даты, но выдается моделью наружу в формате, соответствующем текущей локализации.

Виртуальные поля – это поля, которые физически не существуют в БД, как отдельное поле, но возвращаются моделью. Пример виртуального поля – поле «возраст», которое вычисляется «на лету» в зависимости от даты рождения, и которое не нужно сохранять в БД.

Виртуальные поля являются частным видом вычисляемых полей, поэтому далее будет идти речь только о вычисляемых полях.

Для того чтобы обозначить поле, как вычисляемое, в конструкторе модели элемента необходимо вызвать метод [setFieldCallback](#). Пример:

```
$this->setFieldCallback('birth', array($this, 'fcbDate'));
```

Первый параметр – название свойства модели, второй – [callback](#). Существует ряд стандартных callback-функций, [описанных в API](#) (), они начинаются с префикса fcb. Пример использования из документации:

```
// AmiSample_TableItem::__construct()
$this->setFieldCallback('birth', array($this, 'fcbDate'));
```

Существует возможность добавления собственных callback-функций. Пример добавления и прототип для собственных callback-функций приведен в описании метода [setFieldCallback \(\)](#). Рекомендуется для собственных callback-функций использовать префикс fcb.

## 7.6 Валидаторы

Валидатор – это метод проверки корректности данных при сохранении модели элемента. Если хотя бы один из валидаторов данных модели возвращает ошибку, модель элемента не может быть сохранена, в этом случае генерируется исключение (exception) AMI\_ModTableItemException ([http://manual-v5.cmspanel.net/docs/api6/ModuleComponent/Model/AMI\\_ModTableItemException.html](http://manual-v5.cmspanel.net/docs/api6/ModuleComponent/Model/AMI_ModTableItemException.html))

Добавление валидаторов осуществляется посредством вызова метода [addValidators](#) . Пример:

```
$this->oTable->addValidators(
    array(
        'header' => array('filled', 'virtual_field_presence'),
        'body' => array('required')
    )
);
```

Существуют системные валидаторы:

Группа	Валидатор
Числовые	'int', 'float', 'double'
Символьные	'char', 'varchar'
Текстовые	'tinytext', 'text', 'mediumtext'
Бинарные данные	'tinyblob', 'blob', 'mediumblob'
Дата	'datetime', 'date', 'time'
Обязательные поля	'required', 'filled'

Поля всех групп, кроме обязательных полей, проверяются на значения и длину, соответствующие аналогичным полям в БД mysql.

Валидатор 'required' отличается от 'filled' тем, что 'required' требует наличие поля, а 'filled' требует не только наличие, но и неравенство значения поля пустой строке.

Существует возможность создание собственных валидаторов. Для добавления собственного валидатора, для определенности назовем валидатор 'virtual\_field\_presence', необходимо:

1. Добавить созданный валидатор обычным способом посредством вызова addValidators
2. Зарегистрировать обработчик события ['on\\_save\\_validate {virtual field presence}'](#)

## 3. Имплементировать код обработчика события

Пример:

```
class DemoModule_TableItem extends AMI_ModTableItem{
    public function __construct(AMI_ModTable $oTable, DB_Query $oQuery = null){
        parent::__construct($oTable, $oQuery);

        // Добавляем валидатор
        $this->oTable->addValidators(
            array(
                'header' => array('virtual_field_presence'),
            )
        );

        // Обработчик события
        AMI_Event::addHandler('on_save_validate_{virtual_field_presence}', array($this,
'validateVirtualFieldPresence'), $this->getModId());
    }

    public function validateVirtualFieldPresence($name, array $aEvent, $handlerModId, $srcModId){
        // Код валидатора
        return $aEvent;
    }
}
```

Параметры события \$aEvent:

Название	Значение	Комментарий
field	Название поля	
value	Значение поля	
oTableItem	Модель валидируемого элемента	Объект класса, унаследованного от AMI_ModTableItem
message	Текст ошибки	

Для того чтобы указать, что валидация не пройдена, необходимо задать непустое значение `$aEvent['message']`.

Валидаторы рекомендуется добавлять в конструкторе модели элемента, т.е. только тогда, когда они действительно могут использоваться.

## Ресурсы

Для того чтобы система имела доступ к разработанным классам, необходимо описать их как ресурсы.

Ресурс позволяет унифицировать доступ к конечному функционалу модулей без знания имен классов.

Например, существует некоторый класс `MyNewsModel`, реализующий функционал модели данных модуля `my_news`, разработанного пользователем. Если класс `MyNewsModel` описан, как ресурс `my_news/table/model`, то для получения модели, имея только имя модуля `my_news`, достаточно получить ресурс модели:

```
AMI::getResourceModel($modId . '/table/model');
```

Этот механизм позволяет разрабатываемым классам взаимодействовать друг с другом через стандартные интерфейсы, без знания имен классов.

Также такой механизм удобен тем, что позволяет изменять имена классов, реализующих функционал без изменения кода вызова данного класса.

### 8.1 Существующие ресурсы.

Перечень доступных ресурсов описан в разделе [Resource List \[English\]](#).

## Компоненты

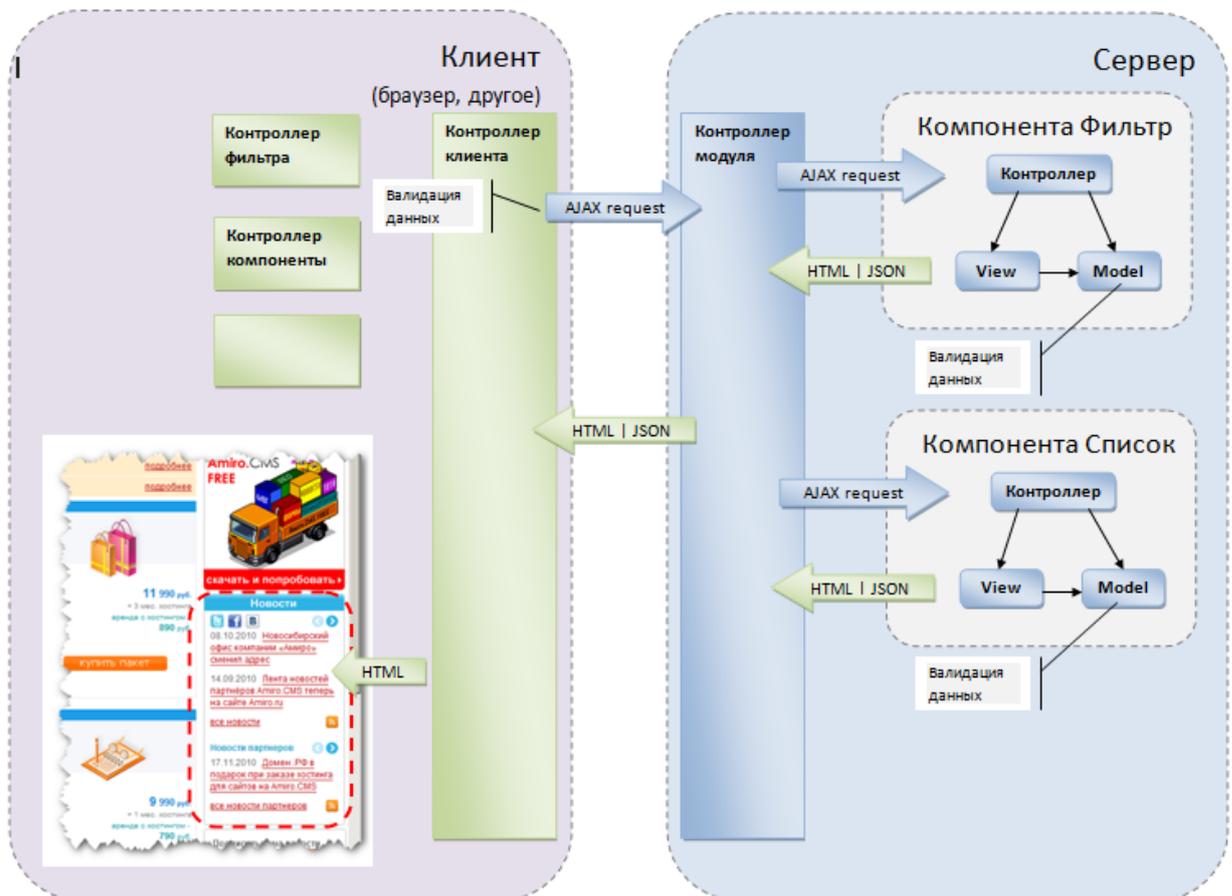
### Общая информация

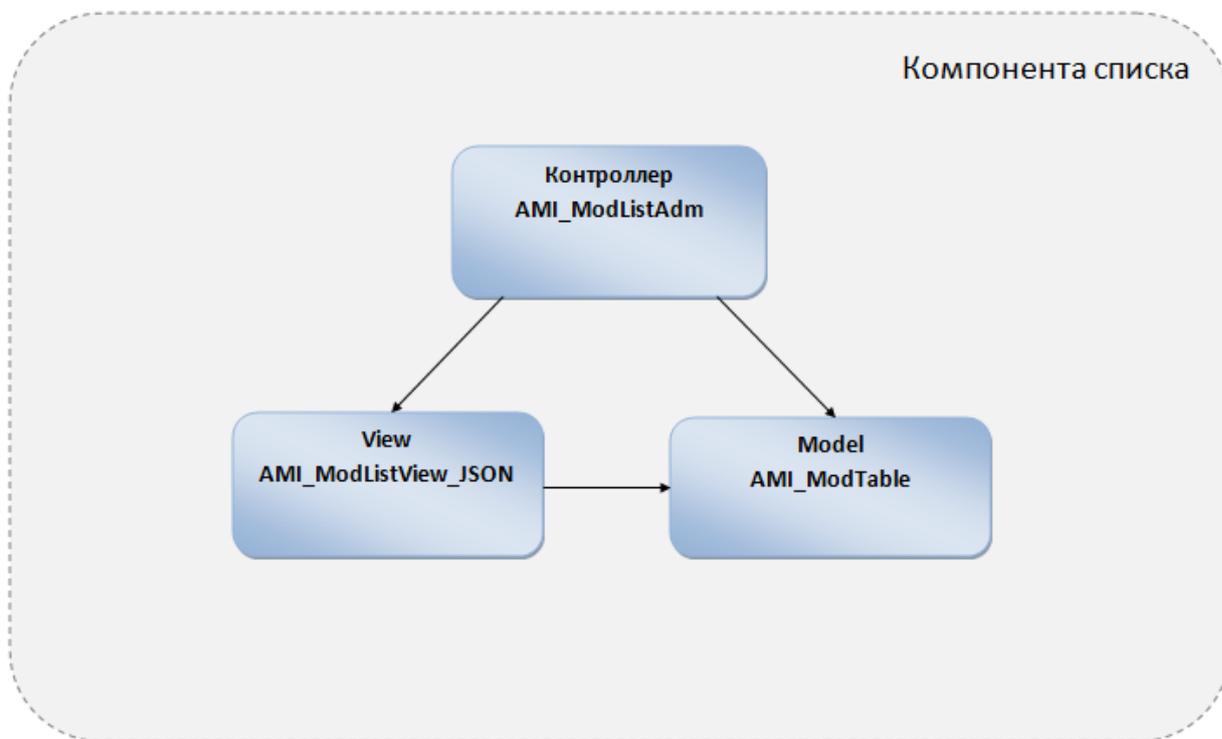
Компонента – это неделимая с точки зрения бизнес-логики и отображения группа элементов в панели управления, предназначенных для выполнения определенных действий. Примеры компонент: список, форма, фильтр.

Компонента состоит из 3 частей (на примере списка):

1. Контроллер ([AMI\\_ModListAdm](#))
2. Модель ([AMI\\_ModTable](#); отображение данных происходит с использованием модели списка [AMI\\_ModTableList](#)).
3. Отображение ([AMI\\_ModListView\\_JSON](#))

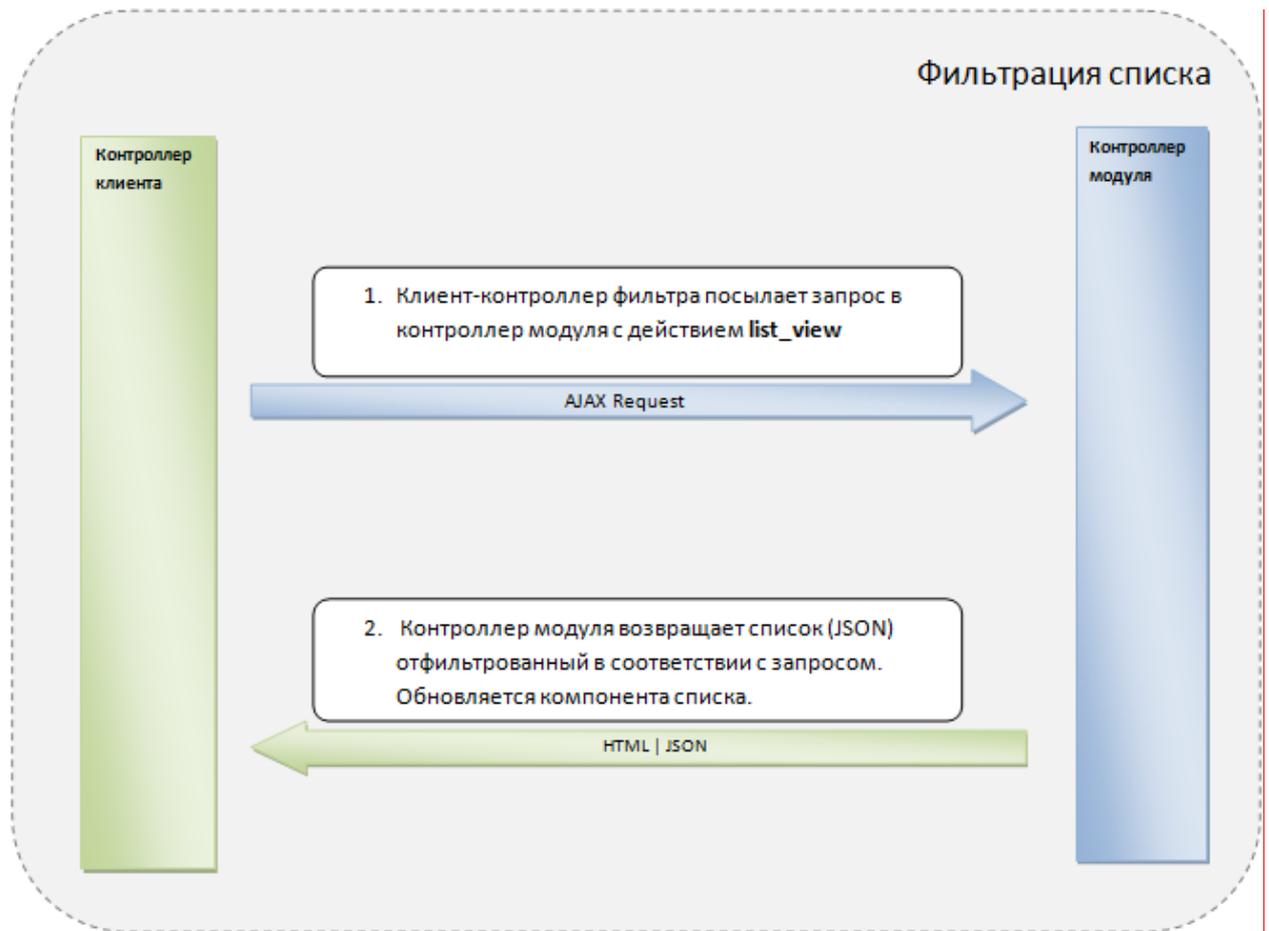
### Схема работы компонент

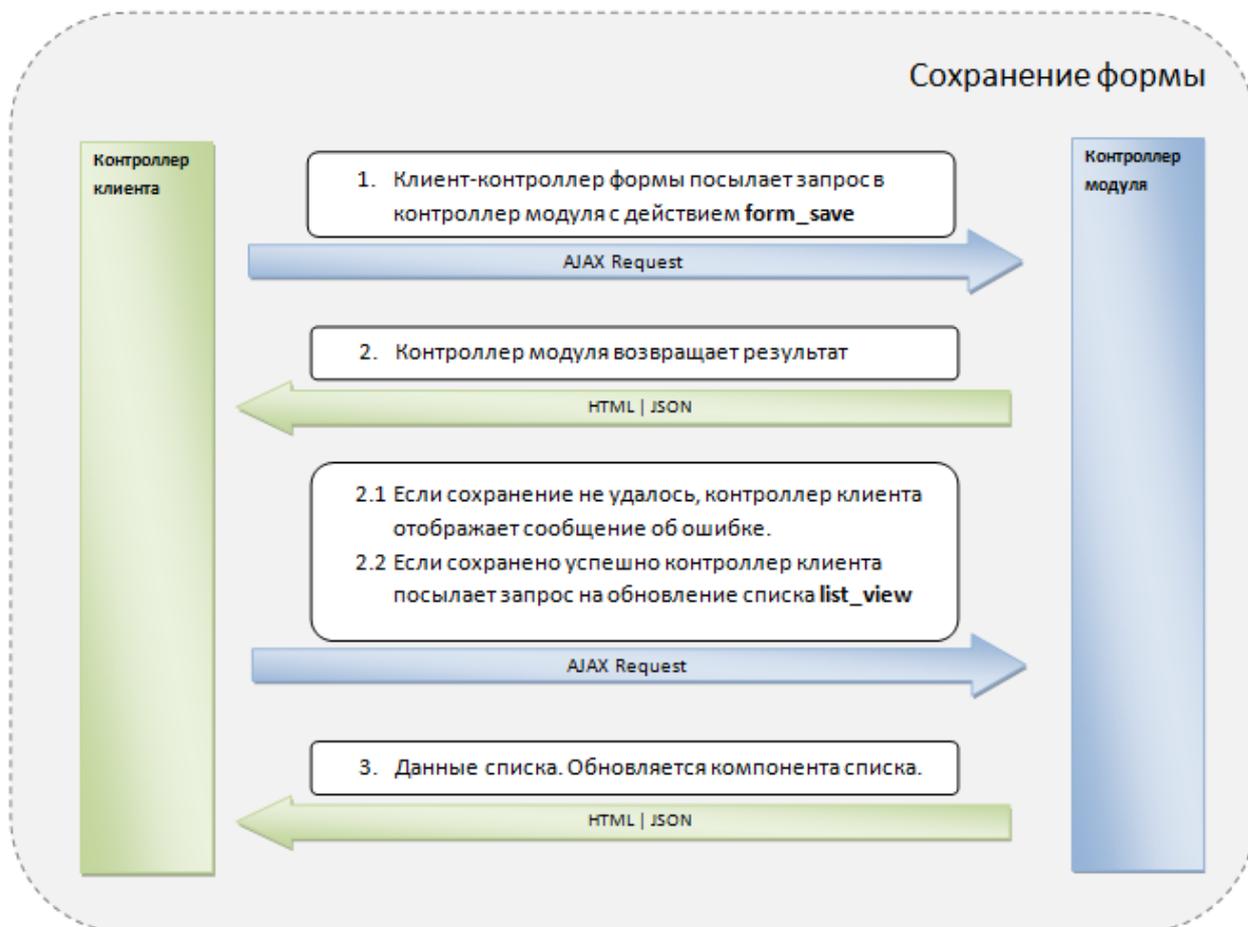




В отличие от публичной части сайта, в панели администратора клиентская часть компоненты и серверная часть компоненты взаимодействуют друг с другом при помощи AJAX технологии. Возможно взаимодействие клиентской части компоненты с серверной, как в HTML, так и JSON формате.

#### Схема взаимодействия контроллеров в панели администратора





Для того чтобы использовать компоненты, необходимо создать [плагин 6.0](#).

Внимание! Во всех последующих пунктах, рассматривающих создание компонент, для простоты будем считать, что создаются компоненты для плагина `ami_sample`, входящего в дистрибутив, и предполагается, что все модели и определения ресурсов уже [созданы](#).

### Табличный список

Компонента списка поддерживает следующие возможности:

1. Отображение таблицы со списком элементов
2. Форматирование полей элементов при отрисовке
3. Поддержка пагинации
4. Установка количества отображаемых элементов на странице
5. Управление отображаемыми столбцами
6. Управление сортировкой столбцов
7. Поддержка действий (стандартные, собственные, групповые операции)

### Создание списка

Для создания компоненты списка необходимо реализовать:

1. Класс контроллера, наследуемый от AMI\_ModListAdm
2. Класс отображения, наследуемый от AMI\_ModListView\_JSON

Пример классов компонент списка:

```
class AmiSample_ListAdm extends AMI_ModListAdm{
}

class AmiSample_ListViewAdm extends AMI_ModListView_JSON{
    public function __construct(){
        $this->tplFileName = '_local/plugins_distr/' . $this->getModId() . '/templates/list.tpl';
        parent::__construct();

        // Init columns
        $this
            ->addColumnType('id', 'hidden')
            ->addColumn('nickname')
            ->addColumn('birth')
            ->addColumnType('age', 'int')
            ->setColumnTensility('nickname');
    }
    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() . '/templates/list.lng';
    }
}
```

Описание методов, использованных в примере, будет приведено ниже.

### **Определение списка и порядка полей**

Метод [addColumn](#) используется для добавления полей в список.

### **Позиции**

Для задания порядка полей используются позиции (placeholders).

Позиция – это место в последовательности, относительно которого могут быть размещены

элементы управления и другие позиции. Механизм применяется для того, чтобы создать единообразие расположения элементов управления.

Позиции бывают 2 видов: позиции элементов и секции. Секцию следует понимать, как логическую группу элементов. Секция отличается от позиции элемента тем, что имеет начало и конец.

Для того чтобы задать порядок полей, необходимо задать порядок позиций при помощи метода [putPlaceholder](#) или [addPlaceholders](#). Последний метод позволяет производить групповое добавление позиций элементов.

Параметры метода `putPlaceholder`:

- название позиции
- место размещения относительно других позиций
- является ли позиция секцией

Пример:

```
$this->putPlaceholder('nickname', 'datefrom.before | seo.begin | form.end');
```

В данном примере добавляется позиция `nickname` по следующему правилу:

- Если есть позиция `datefrom`, то новую позицию добавляют перед ней
- Если позиции `datefrom` нет, но есть секция `seo`, то добавляем позицию в ее начало
- Если ни позиции `datefrom` нет и нет секции `seo`, то добавляем позицию в конце формы

Рассмотрим подробнее возможные относительные положения.

У позиции элемента есть 2 относительных положения для добавления:

- перед позицией элемента (`before`)
- после позиции элемента (`after`)

В отличие от позиции элемента, для секции существуют 4 относительных положения для добавления:

- перед началом секции (`before`)
- в начале секции (`start`)
- в конце секции (`end`)
- после секции (`after`)

Следует понимать:

- позиция может быть не используемой, т.е. ей не будет соответствовать элемента управления
- имя позиции и секции уникально; при добавлении позиции с именем, которое уже есть в списке позиций, она будет добавлена

## Форматтеры

Форматтер – это метод, преобразовывающий данные из модели к виду, требуемому для отображения.

Для того чтобы назначить полю списка форматтер, при формировании списка отображаемых полей, необходимо вызвать метод [formatColumn](#). Пример:

```
$this->formatColumn('nickname', array($this, 'fmtTruncate'));
```

Первый параметр – название поля модели, второй – [callback](#). В данный момент существует один стандартный callback-метод, описанный в [API](#), он начинается с префикса fmt: [fmtTruncate](#).

Существует возможность добавления собственных форматтеров. Пример добавления и прототип для собственных форматтеров приведен в описании метода [formatColumn](#). Рекомендуется для собственных форматтеров также использовать префикс fmt.

## Действия

В список можно добавлять действия над элементами. Действия разделяются по типу отображения и требованию к окружению (действия, модифицирующие данные моделей должны требовать полное окружение, см. пример реализации):

1. Действия в столбце действий (последний в списке);
2. Действия в существующем столбце с данными;
3. Действия в отдельных столбцах;
4. Действия в отдельных столбцах, иконка, всплывающая подсказка и параметры, передаваемые серверу которых зависят от значения поля типа «флаг».

Действия добавляются следующим образом:

1. В контроллере списка (наследнике [AMI\\_ModListAdm](#)) вызываются методы добавления действий;
2. Описываются обработчики действий в контроллере действий списка (наследнике [AMI\\_ModListActions](#));
3. Заполняются данные для локализации в файле «templates/list.lng»;

Для отображения графического файла иконки в списке нужно в папку «img» дистрибутива плагина добавить png-файл с названием «icon-{\$action}.png».

## Групповые действия

В список можно добавлять действия, совершаемые над группой элементов. Для добавления групповой операции необходимо:

1. В контроллере списка (наследнике [AMI\\_ModListAdm](#)) вызываются методы добавления групповых действий;

2. Описать обработчики действий в контроллере действий списка (наследнике [AMI\\_ModListGroupActions](#));
3. Заполнить данные для локализации в файле «templates/client.lng»;
4. Для отображения графического файла иконки в списке нужно папку модуля «img» добавить png-файлы размером 24x24 px с названиями «icon-{\$action}.png», «icon-grp-{\$action}.png».

### Примеры реализации

Задача. Есть модель, содержащая 3 свойства: id, nickname, birth. Необходимо создать классы компонента списка элементов модели, отображающих все 3 эти свойства, nickname необходимо выводить не более чем 50 символов. В списке необходимо иметь следующие возможности: открыть элемент на редактирование, удалить элемент, скопировать элемент.

Должны быть следующие групповые операции: опубликовать, распубликовать, переименовать и удалить.

```
class AmiSample_ListAdm extends AMI_ModListAdm{
    // Id ресурса контроллера действий списка
    protected $listActionsResId = 'ami_sample/list_actions/controller/adm';

    public function init(){
        // AMI_ModListAdm::addActions() must be called before parent::init()!

        // Добавление действий «Редактировать», «Удалить», «Копировать»,
        «Просмотр» в столбец действий (последний в списке), все перечисленные
        действия, кроме действия «Просмотр», модифицируют данные моделей, поэтому
        имеют префикс AMI_ModListAdm::REQUIRE_FULL_ENV

        $this->addActions(array(self::REQUIRE_FULL_ENV . 'edit',
        self::REQUIRE_FULL_ENV . 'delete', self::REQUIRE_FULL_ENV . 'copy', 'show'));

        // Добавление групповых действий «Опубликовать», «Распубликовать»,
        «Переименовать», «Удалить»
        // Первым элементом массива, описывающего каждое действие
        // является имя действия. Все перечисленные действия модифицируют
        // данные моделей, поэтому имеют префикс
        // AMI_ModListAdm::REQUIRE_FULL_ENV
        // Вторым элементом следует позиция действия в списке, механизм
        // аналогичен установке позиций полей в списке. Если позиция
        // встречается впервые и не содержит точки, оно рассматривается как
        // имя секции. Секции разделяются между собой вертикальным
        // разделителем.
```

```

        $this->addGroupActions(
            array(
                array(self::REQUIRE_FULL_ENV . 'public', 'public_section'),
                array(self::REQUIRE_FULL_ENV . 'unpublic', 'public_section'),

                array(self::REQUIRE_FULL_ENV . 'rename', 'rename_section'),
                array(self::REQUIRE_FULL_ENV . 'delete', 'delete_section'));
            parent::init();
            return $this;
        }
    }
}

```

```

class AmiSample_ListActionsAdm extends AMI_ModListActions{
    // Обработчик действия «Копировать»
    public function dispatchCopy($name, array $aEvent, $handlerModId,
        $srcModId){
        $oItem = $this->getItem($this->getRequestId());
        $nickname = $oItem->nickname;
        $newNickname = '- ' . $nickname;
        $oItem->nickname = $newNickname;
        $oItem->resetId();
        $oItem->save();
        $aEvent['oResponse']->addStatusMessage(
            'status_copied',
            array(
                'source'      => $nickname,
                'destination' => $newNickname
            )
        );
        $this->refreshView();
        return $aEvent;
    }
}

```

```

class AmiSample_ListGroupActionsAdm extends AMI_ModListGroupActions{
    // Обработчик группового действия «Удалить»
    public function dispatchGrpDelete($name, array $aEvent, $handlerModId,
        $srcModId){
        return parent::dispatchGrpDelete($name, $aEvent, $handlerModId,

```

```
$srcModId);
    }

    // Обработчик группового действия «Переименовать»
    public function dispatchGrpRename($name, array $aEvent, $handlerModId,
    $srcModId){
        $oListActionsAdm = AMI::getResource($handlerModId .
        '/list_actions/controller/adm');

        $aRequestIds = $this->getRequestIds();
        foreach($aRequestIds as $id){
            // Устанавливаем id элемента для последующего вызова действия
            «Переименовать»
            $aEvent['oRequest']->set('mod_action_id', $id);
            // Сохраняем данные события для последующего использования в
            контроллере действий списка
            $oListActionsAdm->setActionData($name, $aEvent, $handlerModId,
            $srcModId);
            $oListActionsAdm->dispatchRename($name, $aEvent, $handlerModId,
            $srcModId);
        }

        // Сбрасываем все статусные сообщения, которые были добавлены при
        вызове $oListActionsAdm->dispatchRename()
        $aEvent['oResponse']->resetStatusMessages();
        // Задаём статусное сообщение с информацией о количестве
        переименованных элементов
        $aEvent['oResponse']->addStatusMessage('status_grp_rename',
        array('num_items' => sizeof($aRequestIds)));

        // Обновление компоненты списка на клиенте после выполнения действия
        $this->refreshView();

        return $aEvent;
    }
}

class AmiSample_ListViewAdm extends AMI_ModListView_JSON{
```

```
public function __construct(){
    $this->tplFileName = '_local/plugins_distr/' . $this->getModId() .
'/templates/list.tpl';
    parent::__construct();

    // Инициализация столбцов
    $this
        // добавляем столбцы
        ->addColumn('id')
        ->addColumn('nickname')
        ->addColumn('birth')
        // добавляем виртуальный столбец и задаём его тип
        ->addColumnType('age', 'int')
        // задаём автоматическую ширину столбцу «Имя»
        ->setColumnTensility('nickname')
        // задаём столбцы, по которым можно для сортировки
        ->addSortColumns(
            array(
                'public',
                'nickname',
                'birth'
            )
        );

    // Задаём столбцу с действием «Переименовать» место отображения перед
столбцом «Имя»
    $this->putPlaceholder('rename', 'nickname.before');

    // Задаём максимальную длину строки 50 символов для столбца «Имя»
    $this->formatColumn(
        'nickname',
        array($this, 'fmtTruncate'),
        array(
            'length' => 50
        )
    );
};
```

```
    }  
  
    protected function getModLocalePath(){  
        return '_local/plugins_distr/' . $this->getModId() .  
'/templates/list.lng';  
    }  
}
```

Подробное описание логики работы примера в более сложном виде описано в главе [«Пример реализации модуля»](#).

## Форма

Компонента формы поддерживает следующие возможности:

1. Отображение управляющих элементов (поле ввода, текстовое поле, календарь, radio-кнопки, выпадающий список)
2. Отправка введенных данных (сохранение добавляемого или редактируемого элемента)
3. Валидация данных.
4. Обработка результатов отправки данных (успех или ошибка)

## Создание формы

Для создания компоненты формы необходимо реализовать:

1. Класс контроллера, наследуемый от AMI\_ModForm
2. Класс отображения, наследуемый от AMI\_ModFormViewAdm

Пример классов компонент формы:

```
class AmiSample_FormAdm extends AMI_ModFormAdm{  
    /**  
     * Save action dispatcher.  
     *  
     * @param array &$aEvent Event data  
     * @return void  
     */  
    protected function _save(array &$aEvent){  
        parent::_save($aEvent);  
        if(is_object($this->oModelItem)){
```

```
AMI::getSingleton('response')->addStatusMessage('origin_data',
array('data' => d::getDumpAsString($this->oModelItem->getOriginData())));

AMI::getSingleton('response')-
>addStatusMessage('difference_from_origin', array('data' =>
d::getDumpAsString($this->oModelItem->getDiffFromOrigin())));

    }
}
}

class AmiSample_FormViewAdm extends AMI_ModFormViewAdm{
    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() .
'/templates/form.lng';
    }

    public function __construct(){
        parent::__construct();

        $this->addField(array('name' => 'id', 'type' => 'hidden'));
        $this->addField(array('name' => 'nickname'));
        $this->addField(array('name' => 'mod_action', 'value' => 'form_save',
'type' => 'hidden'));
    }
}
```

Описание методов, использованных в примере, будет приведено ниже.

### Определение списка и порядка полей

Для добавления поля на форму необходимо использовать метод `addField` ([http://manual-v5.cmspanel.net/docs/api6/ModuleComponent/View/AMI\\_ModFormView.html#methodaddField](http://manual-v5.cmspanel.net/docs/api6/ModuleComponent/View/AMI_ModFormView.html#methodaddField)).

Пример:

```
$this->addField(array('name' => 'id', 'type' => 'hidden'));

$this->addField(array('name' => 'nickname'));

$this->addField(array('name' => 'birth', 'type' => 'date', 'validate' => array('custom',
'stop_on_error')));

$this->addField(array('name' => 'creation_date', 'type' => 'date', 'display_by_action' => 'edit'));
```

Параметром метода является ассоциативный массив:

Ключ	Значение	Комментарий
name	Строка	Название поля
type	input, checkbox, datetime, date, select, radio, hidden, htmleditor	Тип поля
value	Строка	Текущее значение
data	Массив значений select, radio	Каждый элемент массива является массивом, определяющим 1 значение контрола. Допустимые ключи id, value, caption, checked, disabled
position	Строка	Положение элемента относительно существующих <a href="#">позиций</a>
html	Строка	HTML код элемента управления
cols	Число	Количество столбцов текста для визуального текстового редактора (htmleditor)
rows	Число	Количество строк текста для визуального текстового редактора (htmleditor)
<code>display_by_action</code>	Строка/Массив строк	Отображать данное поле только в случае перечисленных действий ( <code>edit, new, ...</code> )

Порядок элементов формы задается при помощи формирования [позиций элементов](#), аналогично заданию порядка столбцов в табличном списке.

При добавлении поля создается позиция элемента. Если параметр position не указан явно, то позиция элемента добавляется там, где происходило последнее добавление элементов.

Для создания собственного элемента управления, необходимо передать значение html в массиве параметров. Пример:

```
$this->addField(array('name' => 'my_ctrl', 'html' => $ctrlHtml));
```

### Валидация полей формы на стороне клиента

Возможно добавить свою валидацию поля, которая будет выполняться на стороне клиента.

Например можно проверить возраст участника:

```
$this->addField(array('name' => 'birth', 'type' => 'date', 'validate' => array('custom')));
```

В этом случае при проверке поля будет посылаться сообщение «ON\_FORM\_FIELD\_VALIDATE», которое можно перехватить и проверить на правильность. В случае, если проверка не прошла, достаточно установить флаг `error` массива параметров в `'true'`. Например: можно в плагин в файл `{{templates/form.adm.js}}` добавить код:

```
AMI.Message.addListener(
    'ON_FORM_FIELD_VALIDATE',
    function(oParameters){
        if(oParameters.oField.value.substr(-4)>1995){
            alert(AMI.Template.Locale.get('form_validate_birth_too_young'));
            oParameters.error = true;
            return(false);
        }
        return true;
    }
);
```

А в файл `templates/client.lng`:

```
%%form_validate_birth_too_young%en%%
```

```
User too young!
```

```
%%form_validate_birth_too_young%ru%%
```

```
Пользователь слишком молод!
```

И дата рождения пользователя будет проверяться, и в случае неудачи отобразится соответствующее сообщение.

### Вкладки

Имеется возможность вставить на форму переключаемые области, выполненные в виде вкладок. Для этого сначала нужно зарезервировать на форме область под вкладки.

```
$this->addTabContainer('tabset');
```

Параметром является уникальное имя области, необходимое для добавления вкладок в эту область и позиционирования других элементов формы, относительно области вкладок.

При необходимости область вкладок можно позиционировать, задав позицию вторым параметром.

```
$this->addTabContainer('tabset', 'header.after');
```

Добавление вкладки в область вкладок:

```
$this->addTab('tabname1', 'tabset', self::TAB_STATE_ACTIVE);
$this->addTab('tabname2', 'tabset');
```

Первый параметр – уникальное имя вкладки, предназначенный для позиционирования полей внутри вкладки.

Второй параметр – имя области вкладок, в которой данная вкладка размещается.

Третий, необязательный, параметр – исходное состояние вкладки:

AMI\_ModFormView::TAB\_STATE\_ACTIVE – текущая, выбранная вкладка.

AMI\_ModFormView::TAB\_STATE\_NORMAL – обычная неактивная вкладка (по умолчанию).

AMI\_ModFormView::TAB\_STATE\_DISABLED – “выключенная” вкладка.

Для полей формы вкладки можно рассматривать как секции, и добавление поля на вкладку аналогично включению поля в секцию.

```
$this->addField(array('name' => 'myfield1', 'position' => 'tabname1.end'));
```

Для того, чтобы назначить вкладке читаемый заголовок, необходимо в языковом файле формы установить значения переменных `form_tab_имяВкладки`.

```
%%form_tab_tabname1 %n%%
```

```
Tab title
```

```
%%form_tab_tabname1%ru%%
```

```
Заголовок вкладки
```

### Примеры реализации

Задача. Есть модель, содержащая 4 свойства: `id`, `nickname`, `birth`, `about`. Необходимо создать классы компонента формы, отображающих все 4 эти свойства: `id` – скрытое поле, `nickname` – поле ввода, `birth` – календарь, `about` – визуальный редактор. При этом поля `nickname` и `birth` располагаются на одной вкладке формы, а поле `about` – на другой вкладке.

```
class AmiSample_FormAdm extends AMI_ModFormAdm{
    /**
     * Save action dispatcher.
     *
     * @param array &$aEvent Event data
     * @return void
     */
    protected function _save(array &$aEvent){
        parent::_save($aEvent);
        // Выдаем список исходных данных и изменившиеся данные из этого
        списка
        if(is_object($this->oModelItem)){
            AMI::getSingleton('response')->addStatusMessage('origin_data',
            array('data' => d::getDumpAsString($this->oModelItem->getOriginData()));
            AMI::getSingleton('response')-
            >addStatusMessage('difference_from_origin', array('data' =>
            d::getDumpAsString($this->oModelItem->getDiffFromOrigin()));
        }
    }
}

class AmiSample_FormViewAdm extends AMI_ModFormViewAdm{
    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() .

```

```
    '/templates/form.lng';
    }

    public function __construct(){
        parent::__construct();

        $this->addField(array('name' => 'id', 'type' => 'hidden'));

        // Добавляется область вкладки
        $this->addTabContainer ('tabs');

        // Добавляются вкладки
        $this->addTab ('general_tab', 'tabs', self::TAB_STATE_ACTIVE);
        $this->addTab ('about_tab', 'tabs');

        // Добавляются поля во вкладки
        $this->addField(array('name' => 'nickname', 'position' =>
'general_tab.end'));
        $this->addField(array('name' => 'birth', 'type' => 'date', 'position'
=> 'general_tab.end'));
        $this->addField(array('about' => 'birth', 'type' => 'htmleditor',
'cols' => 80, 'rows' => 6, 'position' => 'about_tab.end'));
        $this->addField(array('name' => 'mod_action', 'value' => 'form_save',
'type' => 'hidden'));
    }
}
```

Подробное описание логики работы примера в более сложном виде описано в главе [«Пример реализации плагина»](#).

## Фильтр

Фильтр является частным случаем формы, поэтому он обладает ее возможностями, настроенными специальным образом:

1. Отображение управляющих элементов фильтра (поле ввода, календарь, выпадающий список)
2. Отправка введенных данных (фильтрация элементов компоненты табличного списка)

## Создание фильтра

Для создания компоненты фильтра необходимо реализовать:

1. Класс контроллера, наследуемый от AMI\_ModFilter
2. Класс отображения, наследуемый от AMI\_ModFilterView
3. Класс модели, наследуемый от AMI\_Filter (следует обратить внимание, что фильтр не работает с моделью элемента – у фильтра своя специфическая модель)

Пример классов компонент фильтра:

```
class AmiSample_FilterAdm extends AMI_ModFilter{
}
```

```
class AmiSample_Filter extends AMI_Filter{
    public function __construct(){
        $this->addViewField(
            array(
                'name'    => 'nickname',
                'type'    => 'input',
                'flt_type' => 'text',
                'flt_default' => "",
                'flt_condition' => 'like',
                'flt_column' => 'nickname'
            )
        );
    }
}
```

```
class AmiSample_FilterView extends AMI_ModFilterView{
    public function __construct(){
        parent::__construct();
        $this->addPlaceholders(array('nickname' => 'datefrom.before'));
    }

    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() . '/templates/filter.lng';
    }
}
```

Описание методов, использованных в примере, будет приведено ниже.

### **Определение списка и порядка полей**

Для добавления поля фильтра необходимо использовать метод [addViewField](#). Пример:

```
$this->addViewField(
    array(
```

```

    'name'      => 'nickname',
    'type'      => 'input',
    'flt_type'  => 'text',
    'flt_default' => "",
    'flt_condition' => 'like',
    'flt_column' => 'nickname'
  )
);

```

Параметром метода является ассоциативный массив. В зависимости от типа добавляемого поля, необходимо указывать различные элементы в массиве.

Общие параметры:

Ключ	Значение	Комментарий
name	Строка	Название поля
type	input, checkbox, datetime, date, select, radio, hidden	Тип поля
flt_type	static, hidden, text, date, select, radio, submit, button, checkbox, subcats_search, flagmap, Vsplitter, Hsplitter, Sblock, Fblock, search, sql, timestamp, numeric	Тип обработки поля
value	Строка	Текущее значение
flt_default	Строка	Значение по умолчанию
flt_condition	like = <= >=	Условие фильтрации
flt_column	Строка	Название поля модели, по которому идет фильтрация
html	Строка	HTML код элемента управления

Порядок элементов фильтр задается при помощи формирования позиций элементов, аналогично заданию порядка столбцов в табличном списке.

Для создания собственного элемента управления, необходимо передать значение html в массиве параметров. Пример:

```
$this->addField(array('name' => 'my_ctrl', 'html' => $ctrlHtml));
```

### Примеры реализации

Задача. Есть модель, содержащая 3 свойства: id, nickname, birth. Необходимо создать классы компонента фильтра, отображающих 3 эти поля: nickname – поле ввода (фильтрация по значению), birth – 2 поля календаря (от – до).

```
class AmiSample_FilterAdm extends AMI_ModFilter{  
}
```

```
class AmiSample_Filter extends AMI_Filter{  
    public function __construct(){  
        $this->addViewField(  
            array(  
                'name'     => 'nickname',  
                'type'     => 'input',  
                'flt_type' => 'text',  
                'flt_default' => '',  
                'flt_condition' => 'like',  
                'flt_column' => 'nickname'  
            )  
        );  
        $this->addViewField(  
            array(  
                'name'     => 'datefrom',  
                'type'     => 'datefrom',  
                'flt_type' => 'date',  
                'flt_default' => AMI_Lib_Date::formatUnixTime(AMI_Lib_Date::UTIME_MIN),  
                'flt_condition' => '>=',  
                'flt_column' => 'birth'  
            )  
        );  
        $this->addViewField(  
            array(  

```

```
'name'    => 'dateto',
'type'    => 'dateto',
'flt_type' => 'date',
'flt_default' => AMI_Lib_Date::formatUnixTime(AMI_Lib_Date::UTIME_MAX),
'flt_condition' => '<',
'flt_column' => 'birth'
)
);
}
}
```

```
class AmiSample_FilterView extends AMI_ModFilterView{
    public function __construct(){
        parent::__construct();

        // Add admin filter form placeholder for nickname
        $this->addPlaceholders(array('nickname' => 'datefrom.before'));
    }

    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() . '/templates/filter.lng';
    }
}
```

Подробное описание логики работы примера в более сложном виде описано в главе [«Пример реализации плагина»](#).

## Статусные сообщения

Статусные сообщения предназначены для вывода результата последнего совершенного действия или описания ошибки, если она произошла во время действия.

По умолчанию компоненты списка и формы уже содержат статусные сообщения удаления, добавления и изменения элементов. Для отображения статусных сообщений в плагине, нужно

добавить соответствующие переменные в языковые файл «messages.lng»:

%%status\_del%% для компоненты списка, %%status\_add%% и %%status\_apply%% для компоненты формы.

Для добавления статусного сообщения используется метод [AMI\\_Response::addStatusMessage\(\)](#).

При добавлении сообщения вызывается событие on\_add\_status\_message:

*Параметры:*

<b>Параметр</b>	<b>Описание</b>	<b>Пример</b>
moduleId	ID модуля	ami_sample
key	Ключ переменной из языкового шаблона	status_add
type	Тип сообщения	AMI_Response::STATUS_MESSAGE
aParams	Параметры сообщения	array()

Возможно три типа сообщений:

AMI\_Response::STATUS\_MESSAGE – обычное сообщение

AMI\_Response::STATUS\_MESSAGE\_WARNING - предупреждение

AMI\_Response::STATUS\_MESSAGE\_ERROR – ошибка

aParams – массив для замены в языковых переменных параметров значениями (ключ => значение).

Перед отдачей массива сообщений вызывается событие on\_get\_status\_messages:

*Параметры:*

<b>Параметр</b>	<b>Описание</b>
aStatusMessages	Массив содержащий все сообщения, подготовленные для вывода. Каждое сообщение содержит параметры: key – ключ переменной из языкового шаблона, message – локализованное сообщение, type – тип сообщения.

Для удаления всех статусных сообщений используется метод:

[AMI\\_Response::resetStatusMessages\(\)](#)

Эти события позволяют влиять на статусные сообщения.

Рассмотрим добавление своего параметризованного сообщения на примере плагина `ami_sample`.

Сообщение будет выводиться при добавлении новой записи, также в сообщении будет выводиться «Имя» новой записи.

В контроллере модуля `AmiSample_Adm` добавляем обработчик события `on_add_status_message`.

```
public function __construct(AMI_Request $oRequest, AMI_Response $oResponse){
    parent::__construct($oRequest, $oResponse);

    // Загрузка локализации статусных сообщений

    $oResponse->loadStatusMessages('_local/plugins_distr/' . $this->getModId() . '/templates/messages.lng');

    // Добавление компонент фильтра, списка и формы

    $this->addComponents(array('filter', 'list', 'form'));

    // Обработчик события добавления статусного сообщения

    AMI_Event::addHandler('on_add_status_message', array($this, 'handleAddStatusMessage'), $this->getModId());
}
```

В обработчике события заменяем ключ сообщения `'status_add'` на новый `'status_name_add'`. В параметры сообщения добавляем «Имя» добавленной записи.

```
public function handleAddStatusMessage($name, array $aEvent,
    $handlerModId, $srcModId){

    if($aEvent['key'] == 'status_add'){
```

```
        $aEvent['key'] = 'status_name_add';

        $aEvent['aParams'] = array('name' =>
AMI::getSingleton('env/request')->get('nickname'));

    }

    return $aEvent;

}
```

В языковой шаблон «messages.lng» добавляем новую переменную с ключом 'status\_name\_add'. Параметры обрамляются подчёркиваниями:

```
%%status_name_add%en%%
Record with name "_name_" was added.
%%status_name_add%ru%%
Запись с именем "_name_" добавлена.
```

## Пример реализации модуля (MVC подход)

С версии 5.12.0 в плагинах стало доступно использование стандартных компонент модулей (пока что только для интерфейса администратора), для этого в конфигурационный файл плагина нужно добавить параметр  
api\_version = 6.00

В данной главе подразумевается, что разработчик уже знаком с организацией кода плагина, поэтому пример реализации модуля будет дан путем модификации плагина «sample». Инструкция по созданию плагина: <http://manual.amiro.ru/api/plugins/vvedenie>.

**Решаемая задача** — создать административный модуль управления списком людей и спецблок вывода списка на общедоступной части сайта. Информация о человеке: ник, дата рождения, общая информация, контактная информация(email, телефон). В списке необходимо вывести ник, дату рождения и возраст пользователя. В списке должны быть доступны действия «Редактировать», «Удалить», «Копировать», «Переименовать», «Опубликовать/распубликовать»; удаление должно быть запрещено для элемента с id == 1. Должны быть следующие групповые операции: «Опубликовать», «Распубликовать», «Переименовать» и «Удалить». В интерфейсе администратора должна быть доступна фильтрация по имени пользователя и дате рождения.

Далее, если идет речь о модификации файла, указывается путь относительно папки

плагина, «\_local/plugins\_distr/my\_sample» (в свою очередь данный путь указан относительно корневой директории сайта).

Потребуется файлы, содержащие:

1. SQL создания таблицы;
2. Точка входа модуля в панели управления;
3. Контроллер модуля в панели управления;
4. Модель таблиц БД;
5. Компонента табличного списка;
6. Компонента фильтра;
7. Компонента формы;
8. Точка входа модуля для публичной части сайта (спецблок);
9. Контроллер модуля для публичной части сайта;
10. Отображение списка для публичной части сайта;
11. Файлы шаблонов и локализаций.
12. Файлы иконок действий.

Для простоты создания модуля, несколько классов объединяются в один файл. В больших расширяемых проектах классы рекомендуется размещать в отдельных файлах.

Исходя из задачи, делаем следующие шаги до установки модуля:

1. Копируем папку «\_local/plugins\_distr/sample» в «\_local/plugins\_distr/my\_sample»
2. Модифицируем «\_local/plugins\_distr/my\_sample/config.php»:
  1. Изменяем id плагина на «*my\_sample*»;
  2. Изменяем версию плагина на 1.00;
  3. Указываем, что плагин работает с API 6.0, добавляя параметр:  
`api_version = 6.00`
  4. Удаляем параметры «other», «config», «copy\_allowed»;
  5. Модифицируем параметры «specblock», «admin»:  
`specblock = my_sample_specblock.php`  
`admin = my_sample_mapping.php`
  6. В параметре «install\_as» заменяем «sample» на «*my\_sample*»:  
`install_as = plugins::my_sample, modules::news::my_sample`

Полученный результирующий файл «config.php»:

```
<code>
<pre><code>
</pre>
</code>
```

```
api_version = 6.00

; specblock, admin file name (at least one of them should be specified)
specblock = my_sample_specblock.php
admin      = my_sample_mapping.php

; sql install file
sql_install = install.sql

; sql uninstall file
sql_uninstall = uninstall.sql

; icon that will be displayed on start page (optional)
icon = icon_sample.gif

; list of allowed modules where plugin will be installed to (it is
required for now, later it will be optional)
; install_as module will be a real module later - system just checks
for such module existence
; module links are considered as NOT installed for now
install_as = plugins::my_sample, modules::news::my_sample
```

### 3. Удаляем лишние файлы:

В папке «code»:

переименовываем файлы «my\_admin.php» в «my\_sample\_mapping.php», «my\_specblock.php» в «my\_sample\_specblock.php», удаляем php-файлы, кроме переименованных.

### 4. Опциональный шаг. Удаляем папку «options», так как наш модуль не будет иметь настроек.

### 5. Создаем SQL-файл, исполняемый при инсталляции плагина «database/install.sql»:

```
DROP TABLE IF EXISTS `my_sample_plugin`;

CREATE TABLE `my_sample_plugin` (
  `id` int(11) NOT NULL auto_increment,
  `public` tinyint(3) unsigned NOT NULL DEFAULT '1',
  `nickname` varchar(255) NOT NULL,
  `birth` date NOT NULL DEFAULT '0000-00-00',
  `about` TEXT,
  `phone` varchar(16) NOT NULL DEFAULT '',
  `email` varchar(64) NOT NULL DEFAULT '',
  PRIMARY KEY (`id`)
) ENGINE=MyISAM;
```

### 6. Создаем SQL-файл, исполняемый при деинсталляции плагина «database/uninstall.sql»:

```
DROP TABLE IF EXISTS `my_sample_plugin`;
```

### 7. Изменяем точку входа плагина «code/my\_sample\_mapping.php» – создаем ресурсы

модуля.

```
<?php  
  
$modId = 'my_sample';  
  
/*
```

Добавление ресурсов модели данных:

```
*/  
AMI::addModResources($modId, 'table');
```

Добавление ресурсов модели модуля и контроллера интерфейса администратора:

```
*/  
AMI::addModResources($modId, 'module', array('model',  
'controller/adm'));  
  
/*
```

Добавление ресурсов компоненты фильтра:

```
*/  
AMI::addModResources($modId, 'filter/adm');  
  
/*
```

Добавление ресурсов компоненты списка:

```
*/  
AMI::addModResources($modId, 'list/adm');  
  
// Module admin list action controller  
AMI::addModResources($modId, 'list/adm',  
array('list_actions/controller/adm'));  
  
// Module admin list group action controller  
AMI::addModResources($modId, 'list/adm',  
array('list_group_actions/controller/adm'));  
  
/*
```

Добавление ресурсов компоненты формы:

```
*/  
AMI::addModResources($modId, 'form/adm');
```

8. **O** Создаем файл «code/ami\_sample\_mapping\_frn.php» с ресурсами модуля для общедоступной части сайта.

```
<?php
```

```
$modId = 'my_sample';
```

```
/*
```

Добавление ресурса контроллера модуля для общедоступной части сайта:

```
*/
```

```
AMI::addModResources($modId, 'module', array('controller/frn'));
```

```
/*
```

Задание соответствия ресурса компоненты списка имени класса:

```
*/
```

```
AMI::addResourceMapping(
```

```
    array(
```

```
        $modId . '/list/view/frn' => AMI::getClassPrefix($modId) .
```

```
        '_ListViewFrn',
```

```
    )
```

```
);
```

9. **O** Создаем классы контроллера плагина и модели плагина  
«code/MySample\_Adm.php»:

```
<?php
```

```
class MySample_Adm extends AMI_Mod{  
    public function __construct(AMI_Request $oRequest, AMI_Response  
$oResponse){  
        parent::__construct($oRequest, $oResponse);  
    }  
}
```

Добавление компонент фильтра, списка и формы:

```
*/
```

```
    $this->addComponents(array('filter', 'list', 'form'));
```

```
}
```

```
/*
```

Указание файла локализации сообщений интерфейса администратора:

```
*/
```

```
    public function getClientLocalePath(){  
        return '_local/plugins_distr/' . $this->getModId() .  
        '/templates/client.lng';  
    }  
}
```

```
class MySample_State extends AMI_ModState{  
}
```

10. Создаем классы модели данных, элемента и списка  
«code/MySample\_Table.php»:

```
<?php
```

```
class MySample_Table extends AMI_ModTable{  
/*
```

Указание названия таблицы:

```
*/  
    protected $tableName = 'my_sample_plugin';  
}
```

```
class MySample_TableItem extends AMI_ModTableItem{  
    public function __construct(AMI_ModTable $oTable, DB_Query $oQuery  
= null){  
        parent::__construct($oTable, $oQuery);  
  
        // Add field validators
```

```
/*
```

Добавление валидаторов (поля должны быть не пустыми, при ошибке производить остановку дальнейших проверок):

```
*/  
        $this->oTable->addValidators(  
            array(  
                'nickname' => array('filled', 'stop_on_error'),  
                'birth'    => array('filled', 'stop_on_error')  
            )  
        );
```

```
/*
```

Форматировать поле, как дату (стандартный форматтер):

```
*/  
        // Add field callback to process human-readable date from  
request  
        $this->setFieldCallback('birth', array($this, 'fcbDate'));
```

```
/*
```

Добавление собственного форматтера для виртуального поля:

```
*/  
        // Add virtual field callback  
        $this->setFieldCallback('age', array($this, 'fcbAge'));  
    }  
/*
```

Метод-форматтер виртуального поля, возвращающий при получении значения поля вычисляемое значение и не позволяющий сохранить данное поле в базу при

сохранении:

```

*/
    protected function fcbAge(array $aData){
        if($aData['action'] === 'get'){
            // Sample age calculation
            list($y, $m, $d) = explode('-', $this->aData['birth']);
            $aData['value'] = date('Y') - $y - (int)(date('md') < ($m .
$d));
        }else{
            $aData['_skip'] = true;
        }
    }
}

class MySample_TableList extends AMI_ModTableList{
}

```

11. Создаем класс модели модификатора элемента «code/MySample\_TableItemModifier.php»:

```

class AmiSample_TableItemModifier extends AMI_ModTableItemModifier{
}

```

12. Создаем классы контроллера и отображения табличного списка «code/MySample\_ListAdm.php»:

```

<?php

class MySample_ListAdm extends AMI_ModListAdm{
    // List actions controller resource id
    protected $listActionsResId =
'my_sample/list_actions/controller/adm';
    public function init(){
/*

```

Добавление вывода иконок стандартных действий редактирования и удаления, добавление действия «Копировать» в столбец действий в конце списка:

```

*/
        // AMI_ModListAdm::addActions() must be called before
parent::init()!
        // add actions to "actions" column
        $this->addActions(array('edit', 'delete', 'copy'));

/*

```

Добавление действия публикации/распубликации в отдельный столбец, иконка, всплывающая подсказка и действие, передаваемое на сервер будут зависеть от значения поля элемента:

```

*/

```

```

        // add separate column action
        $this->addColActions(array('public'), true);

    /*
    Добавление действия переименования в отдельный столбец:
    */
        // add separate column action
        $this->addColActions(array('rename'));

    /*
    Добавление действия «Действие внутри содержимого ячейки» в существующий
    столбец «Имя»:
    */

        // add action inside existing column
        $this->addActions(array('inner'), 'nickname');

        // add group actions "public", "unpublic", "rename", "delete"
        $this->addGroupActions(
            array(
                array(self::REQUIRE_FULL_ENV . 'public',
'public_section'),
                array(self::REQUIRE_FULL_ENV . 'unpublic',
'public_section'),
                array(self::REQUIRE_FULL_ENV . 'rename',
'rename_section'),
                array(self::REQUIRE_FULL_ENV . 'delete',
'delete_section'));
            parent::__init();
            return $this;
        }
    }

class MySample_ListViewAdm extends AMI_ModListView_JSON{
    public function __construct(){
        parent::__construct();

    /*

    Добавление столбцов «id», «nickname», «birth», виртуального столбца «age» и
    указание растягиваемого столбца «nickname»:

    */
        // Init colimms
        $this
            ->addColumn('id')
            ->addColumn('nickname')
            ->addColumn('birth')
            ->addColumnType('age', 'int')
            ->setColumnTensility('nickname')

    /*
    Задание столбцов, по которым можно сортировать список («Опубликованность»,
    «Имя», «День рождения»), сортировка по виртуальным полям («age», «Возраст»)

```

недопустима:

```
*/
        ->addSortColumns(
            array(
                'public',
                'nickname',
                'birth'
            )
        );
;
/*
```

Поясним логику задания формата столбцов списка.

Если в конструкторе класса-наследника [AMI\\_ModListView\\_JSON](#) в явном виде не вызван метод [AMI\\_ModListView\\_JSON::addColumnType\(\)](#), будут проанализированы поля, добавляемые во View формы вызовами [AMI\\_ModFormView::addField\(\)](#):

- Поля, для которых отсутствует вызов [AMI\\_ModFormView::addField\(\)](#) или у них указан тип «hidden», преобразуются в скрытые столбцы;
- Поля с типами «date», «datetime» преобразуются в отцентрированные столбцы, содержащие текстовые данные (тип «text»);
- Поля с типами «input», «checkbox», «textarea», «select» и «radio» преобразуются в столбцы, содержащие текстовые данные (тип «text»).

Поля, для которых в наследнике [AMI\\_ModListAdm::init\(\)](#) заданы действия вызовом [AMI\\_ModListAdm::addColActions\(\)](#) в виде

```
// AmiSample_ListAdm:: init()
// add separate column action
$this->addColActions(array('public'), true);
```

преобразуются в столбцы с типом «action».

```
*/
```

```
/*
```

Указание места размещения с действием «Переименовать» перед столбцом «Имя»:

```
*/
```

```
// place 'rename' action column before 'nickname' column
$this->putPlaceholder('rename', 'nickname.before');
```

```
/*
```

Форматирование столбца «nickname» при помощи стандартного форматтера `fmtTruncate`:

```
*/
```

```
// Truncate 'header' column by 50 symbols
$this->formatColumn(
    'nickname',
    array($this, 'fmtTruncate'),
```

```

        array(
            'length' => 50
        )
    );

```

```
/*
```

Форматирование даты рождения при помощи собственного форматтера:

```
*/
```

```

    // Use custom formatter on 'value' column
    $this->formatColumn(
        'birth',
        array($this, 'fmtCustom')
    );

```

```
/*
```

Добавление своего обработчика на действия списка «Редактировать», «Копировать».

```
*/
```

```

    // Add custom script
    $this->addScriptFile('_local/plugins_distr/' . $this-
>getModId() . '/templates/list.adm.js');
}

```

```
/*
```

Указание файла локализации компоненты списка:

```
*/
```

```

    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() .
'/templates/list.lng';
    }

```

```
/*
```

Собственный форматтер, демонстрирующий каким образом можно изменять данные:

```
*/
```

```

    protected function fmtCustom($value, array $aArgs){
        return '~ ' . $value . ' ~';
    }
}

```

### 13. Создаем класс контроллера действий списка «code/MySample\_ListActionsAdm.php»:

```
<?php
```

```

class AmiSample_ListActionsAdm extends AMI_ModListActions{
    /**
     * Dispatches 'delete' action.

```

```

*
* Overridden item deletion. Denies deletion of item having id ==
1.
*/
public function dispatchDelete($name, array $aEvent, $handlerModId,
$srcModId){
    $oItem = $this->getItem($this->getRequestId());
    if($oItem->getId() == 1){
        $aEvent['oResponse']-
>addStatusMessage('status_del_firbidden', array(),
AMI_Response::STATUS_MESSAGE_ERROR);
        $this->refreshView();
    }else{
        parent::dispatchDelete($name, $aEvent, $handlerModId,
$srcModId);
    }
    return $aEvent;
}

/**
* Dispatches 'inner' action.
*
* Displays three messages.
*/
public function dispatchInner($name, array $aEvent, $handlerModId,
$srcModId){
    // Метод dispatchInner() обрабатывает действие в столбце «Имя».
    // Обработка заключается в выдаче статусных сообщений трёх
ТИПОВ
    $aEvent['oResponse']
        ->addStatusMessage('status_inner_note')
        ->addStatusMessage('status_inner_warning', array(),
AMI_Response::STATUS_MESSAGE_WARNING)
        ->addStatusMessage('status_inner_error', array(),
AMI_Response::STATUS_MESSAGE_ERROR);
    // и вызове для обновления View компоненты списка
    $this->refreshView();
    return $aEvent;
}

/**
* Dispatches 'rename' action.
*
* Renames 'nickname' field to "[{$nickname}]".
*/
public function dispatchRename($name, array $aEvent, $handlerModId,
$srcModId){
    // Метод dispatchRename() обрабатывает действие
«Переименовать»:
    // загружает модель элемента, обрамляет поле «Имя» квадратными
скобками и сохраняет модель
    $oItem = $this->getItem($this->getRequestId());
    $nickname = $oItem->nickname;
    $oItem->nickname = '[' . $nickname . ']';
    $oItem->save();
    // добавляет статусное сообщение о переименовании
    $aEvent['oResponse']->addStatusMessage(

```

```

        'status_renamed',
        array(
            'source'      => $nickname,
            'destination' => $oItem->nickname
        )
    );
    // обновляет View компоненты списка
    $this->refreshView();

    return $aEvent;
}

/**
 * Dispatches 'copy' action.
 *
 * Copies item and prepends '- ' string to the new 'nickname'
field.
 */
public function dispatchCopy($name, array $aEvent, $handlerModId,
$srcModId){
    // Метод dispatchCopy() обрабатывает действие «Копировать»
    // загружает модель элемента, дописывает вначале поля «Имя»
строку "- ", сбрасывает id (primary key) модели элемента и сохраняет
модель
    $oItem = $this->getItem($this->getRequestId());
    $nickname = $oItem->nickname;
    $newNickname = '- ' . $nickname;
    $oItem->nickname = $newNickname;
    $oItem->resetId();
    $oItem->save();
    // добавляет статусное сообщение о копировании
    $aEvent['oResponse']->addStatusMessage(
        'status_copied',
        array(
            'source'      => $nickname,
            'destination' => $newNickname
        )
    );
    // обновляет View компоненты списка
    $this->refreshView();

    return $aEvent;
}

/**#@-*/
}

```

#### 14. Создаем класс контроллера групповых действий списка «code/MySample\_ListGroupActionsAdm.php»:

```

class AmiSample_ListGroupActionsAdm extends AMI_ModListGroupActions{
    /**
     * Dispatches 'delete' group action.
     *
     * Delete group items.
     */
}

```

```

        public function dispatchGrpDelete($name, array $aEvent,
        $handlerModId, $srcModId){
            // Метод dispatchGrpDelete() вызывает групповое удаление
            элементов, реализованное в родительском классе
            return parent::dispatchGrpDelete($name, $aEvent, $handlerModId,
            $srcModId);
        }

        /**
         * Dispatches 'rename' action.
         *
         * Rename group items.
         */
        public function dispatchGrpRename($name, array $aEvent,
        $handlerModId, $srcModId){
            // Метод dispatchGrpRename () вызывает обработчик
            dispatchRename, реализованный в контроллере действий списка, для
            каждого элемента группы
            // Создаём объект контроллера действий списка
            $oListActionsAdm = AMI::getResource($handlerModId .
            '/list_actions/controller/adm');

            $aRequestIds = $this->getRequestIds();
            foreach($aRequestIds as $id){
                $aEvent['oRequest']->set('mod_action_id', $id);
                $oListActionsAdm->setActionData($name, $aEvent,
                $handlerModId, $srcModId);
                $oListActionsAdm->dispatchRename($name, $aEvent,
                $handlerModId, $srcModId);
            }

            // Удаляем статусные сообщения, возникшие при вызове
            $oListActionsAdm->dispatchRename() и выдаём общее сообщение
            $aEvent['oResponse']->resetStatusMessages();
            $aEvent['oResponse']->addStatusMessage('status_grp_rename',
            array('num_items' => $itemsProcessed));

            $this->refreshView();

            return $aEvent;
        }
    }
}

```

## 15. Создаем классы контроллера, модели и отображения фильтра «code/MySample\_FilterAdm.php»:

```

<?php

// Контроллер компоненты фильтра
class MySample_FilterAdm extends AMI_ModFilter{
}

// Модель компоненты фильтра, поля добавляются в модели, так как при
фильтрации списка элементов View фильтра не используется и не может
содержать описания полей

```

```

class MySample_FilterModelAdm extends AMI_Filter{
    public function __construct(){
        // Добавление полей «Имя», «С даты», «По дату» в фильтр
        $this->addViewField(
            array(
                'name'          => 'nickname',
                'type'          => 'input',
                'flt_type'      => 'text',
                'flt_default'   => '',
                'flt_condition' => 'like',
                'flt_column'    => 'nickname'
            )
        );
        $this->addViewField(
            array(
                'name'          => 'datefrom',
                'type'          => 'datefrom',
                'flt_type'      => 'date',
                'flt_default'   =>
AMI_Lib_Date::formatUnixTime(AMI_Lib_Date::UTIME_MIN),
                'flt_condition' => '>=',
                'flt_column'    => 'birth'
            )
        );
        $this->addViewField(
            array(
                'name'          => 'dateto',
                'type'          => 'dateto',
                'flt_type'      => 'date',
                'flt_default'   =>
AMI_Lib_Date::formatUnixTime(AMI_Lib_Date::UTIME_MAX),
                'flt_condition' => '<',
                'flt_column'    => 'birth'
            )
        );
    }
}

class MySample_FilterViewAdm extends AMI_ModFilterView{
    public function __construct(){
        parent::__construct();

        // Указание следования поля «Имя» перед полем «С даты»
        $this->putPlaceholder('nickname', 'datefrom.before');
    }

    // Указание файла локализации компоненты фильтра
    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() .
'/templates/filter.lng';
    }
}

```

## 16. Создаем классы контроллера и отображения формы

«code/MySample\_FormAdm.php»:

```
<?php

class MySample_FormAdm extends AMI_ModForm{
}

class MySample_FormViewAdm extends AMI_ModFormViewAdm{
    public function __construct(){
        parent::__construct();

        // Добавление общих полей («id», кнопки сохранения)
        $this->addField(array('name' => 'id', 'type' => 'hidden'));
        // Form save action
        $this->addField(array('name' => 'mod_action', 'value' =>
'form_save', 'type' => 'hidden'));

        // Добавление полей «Публиковать», «Имя», «День рождения»
        $this->addField(array('name' => 'public', 'type' => 'checkbox',
'default_checked' => true));
        $this->addField(array('name' => 'nickname'));
        $this->addField(array('name' => 'birth', 'type' => 'date'));

        // Создание области вкладок «tabset», добавление вкладок
«Информация» и «Контакты»
        $this->addTabContainer('tabset');
        $this->addTab('info', 'tabset', 'active');
        $this->addTab('contacts', 'tabset');

        // Добавление поля на вкладку «Информация», полей «Телефон» и
«Email» на вкладку «Контакты»
        $this->addField(array('name' => 'about', 'type' =>
'htmleditor', 'cols' => 80, 'rows' => 10, 'position' => 'info.end'));
        $this->addField(array('name' => 'phone', 'position' =>
'contacts.end'));
        $this->addField(array('name' => 'email', 'position' =>
'contacts.end', 'validators' => 'email'));
    }

    // Указание файла локализации компоненты формы
    protected function getModLocalePath(){
        return '_local/plugins_distr/' . $this->getModId() .
'/templates/form.lng';
    }
}
```

В папке «templates» удаляем старые и создаем новые файлы локализации панели управления плагина (в кодировке UTF-8):

Файл «templates/messages.lng»:

```
%%status_add%en%%
Record was added.
%%status_add%ru%%
Запись добавлена.

%%status_add_fail%en%%
```

Record was not added!  
%%status\_add\_fail%ru%%  
Запись не добавлена!

%%status\_apply%en%%  
Record was updated.  
%%status\_apply%ru%%  
Запись изменена.

%%status\_apply\_fail%en%%  
Record was not updated!  
%%status\_apply\_fail%ru%%  
Запись не изменена!

%%status\_name\_add%en%%  
Record with name '\_name\_' was added.  
%%status\_name\_add%ru%%  
Запись с именем '\_name\_' добавлена.

%%status\_renamed%en%%  
Record "\_source\_" was renamed to "\_destination".  
%%status\_renamed%ru%%  
Запись "\_source\_" переименована в "\_destination".

%%status\_del%en%%  
Record was deleted.  
%%status\_del%ru%%  
Запись удалена.

%%status\_del\_fail%en%%  
Record was not deleted.  
%%status\_del\_fail%ru%%  
Запись не удалена.

%%status\_del\_firbidden%en%%  
Record deletion is firbidden.  
%%status\_del\_firbidden%ru%%  
Удаление записи запрещено.

%%status\_grp\_del%en%%  
\_num\_items\_ item was deleted.  
%%status\_grp\_del%ru%%  
Удалено элементов: \_num\_items\_.

%%status\_grp\_rename%en%%  
\_num\_items\_ item was renamed.  
%%status\_grp\_rename%ru%%  
Переименовано элементов: \_num\_items\_.

%%status\_copied%en%%  
Record "\_source\_" was copied to "\_destination".  
%%status\_copied%ru%%  
Запись "\_source\_" скопирована в "\_destination".

%%status\_inner\_note%en%%  
Inner cell action note.  
%%status\_inner\_note%ru%%  
Сообщение действия внутри содержимого ячейки.

```
%%status_inner_warning%en%%  
Inner cell action warning.  
%%status_inner_warning%ru%%  
Предупреждение действия внутри содержимого ячейки.
```

```
%%status_inner_error%en%%  
Inner cell action error.  
%%status_inner_error%ru%%  
Ошибка действия внутри содержимого ячейки.
```

#### Файл «templates/client.lng»:

```
##-- list { --##  
  
%%list_edit_action_called%en%%  
Edit action is called...  
%%list_edit_action_called%ru%%  
Открытие элемента на редактирование...  
  
%%list_confirm_copy%en%%  
Are you sure to copy record?  
%%list_confirm_copy%ru%%  
Скопировать запись?  
  
##-- } list --##  
  
##-- group action { --##  
  
%%list_action_grp_rename%en%%  
Rename  
%%list_action_grp_rename%ru%%  
Переименовать  
  
##-- } group action --##
```

#### Файл «templates/list.lng»:

```
%%include_language  
"_local/plugins_distr/ami_sample/templates/common.lng"%%  
  
##-- list column headers { --##  
  
%%list_col_rename%en%%  
%%list_col_rename%ru%%  
  
%%list_col_age%en%%  
Age  
%%list_col_age%ru%%  
Возраст  
  
%%list_col_rename%en%%  
%%list_col_rename%ru%%  
  
##-- } list column headers --##  
##-- list actions { --##
```

```
%%list_action_public%en%%
Published on site, click to make unpublished
%%list_action_public%ru%%
Опубликовано на сайте

%%list_action_unpublic%en%%
Not published on site, click to publish
%%list_action_unpublic%ru%%
Не опубликовано на сайте

%%list_action_edit%en%%
Edit (Common image and title are overridden, click event handling is
implemented on client side)
%%list_action_edit%ru%%
Редактировать (Стандартные изображение и название изменены, осуществлѐн
перехват обработки события нажатия на стороне клиента)

%%list_action_delete%en%%
Delete
%%list_action_delete%ru%%
Удалить

%%list_action_rename%en%%
Rename
%%list_action_rename%ru%%
Переименовать

%%list_action_copy%en%%
Copy
%%list_action_copy%ru%%
Копировать

%%list_action_inner%en%%
&bull;&bull;&raquo;
%%list_action_inner%ru%%
&bull;&bull;&raquo;

##-- inner action hint { --##
%%list_action_title_inner%en%%
Inner cell action
%%list_action_title_inner%ru%%
Действие внутри содержимого ячейки
##-- } inner action hint --##

##-- } list actions --##
```

Файл «templates/filter.lng»:

```
%%filter_field_nickname%en%%
Nickname:
%%filter_field_nickname%ru%%
Имя:
```

Файл «templates/form.lng»:

```
%%include_language
"_local/plugins_distr/ami_sample/templates/common.lng"%%

##-- form title { --##

%%my_sample_add%en%%
Add record
%%my_sample_add%ru%%
Добавить запись

%%my_sample_apply%en%%
Edit record
%%my_sample_apply%ru%%
Редактировать запись

##-- } form title --##
##-- tabs { --##

%%form_tab_info%en%%
About
%%form_tab_info%ru%%
Информация

%%form_tab_contacts%en%%
Contacts
%%form_tab_contacts%ru%%
Контакты

##-- } tabs --##
##-- fields { --##

%%form_field_birth%en%%
Birthday
%%form_field_birth%ru%%
День рождения

%%form_field_about%en%%
About
%%form_field_about%ru%%
Описание

%%form_field_email%en%%
Email
%%form_field_email%ru%%
Email

%%form_field_phone%en%%
Phone
%%form_field_phone%ru%%
Телефон

##-- } fields --##
```

17. Размещаем в папке «img» gif/png-файлы с названиями «icon-{\$action}.gif» или «icon-{\$action}.png».

На время разработки, чтобы не проводить инсталляцию/деинсталляцию модуля, файлы с иконками можно размещать в папке «\_local/\_admin/images/my\_sample»,

куда они попадают после инсталляции модуля.

#### 18. Модифицируем файл спецблока «code/my\_sample\_specblock.php»:

```
<?php

$modId = 'ami_sample';

/**
 * Подключение описания ресурсов в случае несоответствия именования
 ресурсов/классов общей схемы ресурсов API Reference
 */
/*
// Подключение описания ресурсов
require_once $pluginParams['code_path'] . 'ami_sample_mapping.php';
// Подключение описания ресурсов общедоступной части сайта
require_once $pluginParams['code_path'] . 'ami_sample_mapping_frn.php';
*/

// Создание и инициализация контроллера спецблока модуля массивом
 параметров $pluginParams
 $plugin = AMI::getResource($modId . '/module/controller/frn',
 array($pluginParams));

// Получение результата и возврат его в качестве требуемой переменной
 $resultHtml для дальнейшего вывода результата в качестве спецблока
 $resultHtml = $plugin->getResponse();
```

#### 19. Создаем классы контроллера спецблока плагина «code/MySample\_Frn.php»

```
<?php

class MySample_Frn{

    protected $oView;

    public function __construct(array $aPluginParams){
        // Получение объекта отображения
        $this->oView = AMI::getResource($this->getModId() .
'/list/view/frn');
        // Установка пространства переменных объекта отображения
        $this->oView->setScope(
            array(
                'templates_path' => $aPluginParams['templates_path']
            )
        );

        // Получение модели списка
        $oModelList = AMI::getResourceModel($this->getModId() .
'/table')->getList();

        // Задание модели списка столбцов, требуемых для получения из
        БД
        $oModelList->addColumns(array('id', 'nickname', 'birth'));

        // Передача отображению инициализированной модели
```

```

        $this->oView->setModel($oModelList);
    }

    // Метод получения результатов работы отображения
    public function getResponse(){
        return $this->oView->get();
    }

    // Метод получения идентификатора текущего модуля (метод необходим,
    поскольку класс пока что не наследуется от другого класса, реализующего
    данный метод)
    protected function getModId(){
        return AMI::getModId(get_class($this));
    }
}

```

## 20. Создать класс отображения «code/MySample\_ListViewFrn.php»:

```
<?php
```

```

class MySample_ListViewFrn extends AMI_View{

    protected $tplBlockName = 'my_sample';

    protected $oModel;

    public function get(){
        $res = '';
        // Инициализация шаблонизатора
        $oTpl = $this->getTemplate();

        // Добавление фильтрации по опубликованности и загрузка модели,
        инициализированной контроллером
        $this->oModel->addWhereDef('AND `public` = 1')->load();
        if($this->oModel->count() > 0){
            // Добавление блока шаблона
            $oTpl->addBlock($this->tplBlockName, $this-
            >aScope['templates_path'] . 'front.tpl');
            $rows = '';

            // Итерирование по модели списка и формирование отображения
            моделей элементов
            foreach($this->oModel as $oModelItem){
                $aRowData = array(
                    'id'          => $oModelItem->id,
                    'nickname'   => AMI_Lib_String::htmlChars($oModelItem-
                    >nickname),
                    'birth'      => $oModelItem->birth
                );
                $rows .= $oTpl->parse($this->tplBlockName . ':item_row',
                $aRowData);
            }

            // Формирование полного отображения таблицы
            $res = $oTpl->parse($this->tplBlockName . ':items',
            array('rows' => $rows));
        }else{

```

```

        // No items found
    }
    return $res;
}

// Метод предназначен для типизации передаваемой модели (отображение
работает только с наследниками модели списка AMI_ModTableList)
protected function _setModel(AMI_ModTableList $oModel){
    return parent::_setModel($oModel);
}
}

```

## 21. Создаем файлы шаблона и локализации спецблока плагина

Файл шаблона «templates/front.tpl»:

```

%%include_language
"_local/plugins_distr/my_sample/templates/front.lng"%%

<!--#set var="items" value="
<table border="1" cellpadding=5 cellspacing=2>
  <tr><th>%%id%%</th><th>%%name%%</th><th>%%birth%%</th></tr>
  ##rows##
</table>
"-->

<!--#set var="item_row" value="
<tr>
  <td>##id##</td>
  <td>##nickname##</td>
  <td>##birth##</td>
</tr>
"-->

```

Файл локализации «templates/front.lng»:

```

%%id%en%%
ID
%%id%ru%%
ID

%%name%en%%
Name
%%name%ru%%
Имя

%%birth%en%%
Birthday
%%birth%ru%%

```

## 22. Установить плагин

Устанавливаем модуль через мастер надстроек. Обратите внимание, что система кеширует файл конфигурации плагина, поэтому необходимо для применения изменений в конфигурации, перед тем, как открывать модуль плагинов, увеличить

- версию файла конфигурации. [Подробнее о версии плагина.](#)
23. Добавляем спецблок плагина на какую-либо страницу. В процессе разработки рекомендуется временно отключить кеширование на сайте, [подробнее об отключении кеширования в режиме отладки.](#)
  24. Добавляем данные в модуль в панели управления.
  25. Готово. Можно посмотреть результат работы плагина на сайте.

## События

События позволяют контролировать процесс работы сторонних модулей. Поскольку события вызываются неявно, они создают нелинейность кода, которую можно отследить только через стек вызовов событий и список обработчиков. Поэтому события рекомендуется использовать только тогда, когда невозможен прямой вызов объектов.

Для того, чтобы обработать некоторое событие, необходимо зарегистрировать обработчик события [AMI\\_Event::addHandler](#) и указать, событие какого модуля необходимо обработать.

Для того чтобы передать параметры в дальнейшую цепочку событий, необходимо в обработчике события вернуть массив входящих параметров `$aEvent`. Возвращаемые результаты необходимо добавить в данный возвращаемый параметр.

Для того чтобы остановить дальнейшее выполнение цепочки событий, необходимо в качестве элемента массива вернуть `$aEvent['_break_event'] = true;`

### 10.1 События получения списка элементов

#### 10.1.1 on\_table\_get\_list

Инициализация получения списка объектов. Метод удобно использовать для динамического добавления зависимостей моделей (соединение таблиц).

Параметры:

Параметр	Описание	Пример
modId	Идентификатор модуля	news
oTable	Объект модели таблицы	News_Table

Пример обработчика события. Включение активности у зависимой модели с алиасом «cat»:

```
public function handleTableGetList($name, array $aEvent, $handlerModId,
    $srcModId){

    $oTable = $aEvent[' oTable '];

    if(is_null($oTable->setActiveDependence('cat'))){
```

```

        trigger_error('Categories table not found', E_USER_ERROR);
    }

    return $aEvent;
}

// ...

AMI_Event::addHandler('on_table_get_list', array($this,
'handleTableGetList'), $modId);

```

### 10.1.2 on\_query\_add\_table

Событие, вызываемое при добавлении таблицы в запрос, создаваемый моделью таблицы. Событие вызывается при получении объекта модели списка или модели элемента.

Например, событие возникает при вызове `getList()`, если у модели `oModel` есть зависимые модели:

```
$oModelList = $this->oModel->getList()->addColumns($aColumns)->load();
```

Независимо от того, добавляется таблица основной модели или зависимой, событие имеет одинаковые параметры.

Параметры:

Параметр	Описание	Пример
modId	Идентификатор модуля	news
oQuery	Объект запроса	DB_Query
oTable	Объект модели таблицы	News_Table
alias	Алиас таблицы	cat

Пример обработчика события. Фильтрация результатов по языку, если в таблице есть поле «lang»:

```

public static function handleFilterLangData($name, array $aEvent,
$handlerModId, $srcModId){
    if($aEvent['oTable']->hasField('lang')){
        $aEvent['oQuery']->addWhereDef(

            DB_Query::getSnippetlet('AND %s.`lang` = %s')

```

```

        ->plain($aEvent['alias'])

        ->q(AMI_Registry::get('lang_data'))

    );
}
return $aEvent;
}

// ...

AMI_Event::addHandler(

    'on_query_add_table',

    array('AMI_GlobalFilters', 'handleFilterLangData'),

    AMI_Event::MOD_ANY

);

```

### 10.1.3 on\_query\_add\_joined\_columns

Событие, вызываемое для добавления столбцов в результат запроса, полученного соединением с зависимой моделью.

Например, событие возникает при вызове `getList()`, если у модели `oModel` есть зависимые модели:

```
$oModelList = $this->oModel->getList()->addColumns($aColumns)->load();
```

Параметры:

Параметр	Описание	Пример
modId	Идентификатор модуля	news
oQuery	Объект запроса	DB_Query
oTable	Объект модели таблицы	News_Table
alias	Алиас таблицы	cat
oList	Объект модели списка	News_TableList

Пример обработчика события. Добавление столбцов 'id', 'header' в результат соединения моделей:

```
public function handleAdmQueryAddJoinedColumns($name, array $aEvent,
    $handlerModId, $srcModId){
```

```
$aEvent['oList']->addColumns(array('id', 'header'));  
  
return $aEvent;  
  
}  
  
// ...  
  
AMI_Event::addHandler('on_query_add_joined_columns', array($this,  
'handleQueryAddJoinedColumns'), $modId);
```

#### 10.1.4 on\_list\_recordset

Событие, вызываемое непосредственно перед выполнением запроса получения списка элементов.

*Параметры:*

Параметр	Описание	Пример
modId	Идентификатор модуля	news
oQuery	Объект запроса	DB_Query

*Пример обработчика события. Добавление дополнительной фильтрации по категории:*

```
public function handleListRecordset($name, array $aEvent, $handlerModId,  
$srcModId){  
  
    if($this->idCat > 0) {  
  
        $aEvent['oQuery']->addWhereDef(  
  
            DB_Query::getSnippetlet('AND %s.`id` = %s')  
  
            ->plain($this->prefix)  
  
            ->q($this->idCat)  
  
        );  
  
    }  
  
    return $aEvent;  
  
}  
  
// ...  
  
AMI_Event::addHandler('on_list_recordset', array($this,  
'handleListRecordset'), $modId);
```

### 10.1.5 on\_before\_set\_data\_model\_item

Событие, вызываемое при инициализации модели элемента данными записи из базы данных.

Параметры:

Параметр	Описание	Пример
aData	Массив данных, запись из базы данных	
doAppend	Требуется ли добавление, или замена текущего состояния объекта	True
oTableItem	Инициализируемый объект модели элемента	News_TableItem

Вызов данного события обязателен для полноценной инициализации расширений модуля (категории, изображения и т.п.).

### 10.2 on\_html\_meta\_change

Событие, вызываемое при изменении HTML-мета-тегов.

Параметры:

Параметр	Описание	Пример
item	Название атрибута тега <meta>	name
name	Значение атрибута, задающегося в параметре name	robots
Content	Значение атрибута content тега <meta>	noindex, follow

Например, если модуль работает на статичной странице менеджера сайта, получает данные через GET-параметры и нужно, результаты работы модуля индексировались, в обработчике события необходимо проверить, соответствует ли страница, для которой вызвали событие, странице с модулем, и, если да, установить параметр «index» в TRUE:

```
// _local/front_functions.php

function cstOnHTMLMetaChange($name, array $aEvent, $handlerModId, $srcModId){
    if(
        $aEvent['name'] == 'robots' &&
        // номер страницы в менеджере сайта, на которой работает модуль
        AMI_Registry::get('page/id') == 20088
    ){
```

```
        $aEvent['content'] = 'index, follow';
    }

    return $aEvent;
}

AMI_Event::addHandler('on_html_meta_change', 'cstOnHTMLMetaChange',
AMI_Event::MOD_ANY);
```

## Средства отладки и профилирования

### 11.1 Включение отладки

Средства отладки несколько замедляют работу системы и включаются для выбранных IP адресов. Для включения необходимо перейти в модуль «Настройки системы», выбрать раздел «Система» и модуль «Общие настройки». В разделе «Отладка» можно выбрать до пяти IP адресов (текущий адрес для удобства отображается подсказкой) и индивидуально настроить вывод информации по ним.

Настройки позволяют включать и выключать отладочные конструкции (для использования классом отладки и отладки шаблонов). Для отладки шаблонов в параметре «Настройки для IP \*» необходимо выбрать один из пунктов «Парсер шаблонов: отладочные конструкции» или «Парсер шаблонов: полная информация», для возможности использования средства отладки [AMI\\_Debug](#) необходимо выбрать пункт «Парсер шаблонов: полная информация» и включить параметр «Отображать отладочную информацию».

При отключении кэширования или включении отладочной информации к выводу добавляется красная предупреждающая строка. Она видна только для выбранных IP адресов.

Теперь появляется возможность использовать средства отладки [AMI\\_Debug](#), описание и примеры доступны в [полном справочнике](#).

Примеры использования:

Инструкция `d::w('<h1>My debug</h1>')`; выводит строку как есть:

### My debug

Инструкция `d::pr(array(1,2,3), 'my array')`; выводит первый аргумент, используя функцию [print\\_r\(\)](#):

#### my array (1) models.php: 59

```
Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
```

Инструкция [d::vd\(array\(1,2,3\), 'my array'\)](#); выводит первый аргумент, используя функцию [var\\_dump\(\)](#):

#### my array (1) models.php: 59

```
array(3) {
    [0]=>
    int(1)
    [1]=>
    int(2)
    [2]=>
    int(3)
}
```

Инструкция [d::trace\(\)](#); выводит стек вызова:

#### Debug backtrace: models.php:59

File	Line	Caller	Args
E:\_wwwroot\models.php	59	AMI_Debug::trace()	

## 11.2 Профилирование кода и запросов к БД

При разработке кода всегда необходимо использовать средства профилирования, для оценки потребляемых кодом ресурсов и оптимизации работы плагина.

Для профилирования доступны три метода:

а) Инструкция `AMI::getSingleton('response')->displayBench()`; ([AMI\\_Response::displayBench\(\)](#)) сообщает системе, что нужно отображать результаты профилирования:

**»» Warning: debug mode is enabled, 2010-11-26 :21:31 GMT: 2010-11-26 12:21**

**Script total: 0.050174 sec [DB total: 0.018054 sec / queries: 0.017707 sec (6 times) / fetch: 0.000347 sec (31 times)] [PHP total: 0.032120 sec / peak mem usage: 430 912 bytes]**

**Script total** - общее время работы скрипта;

**DB total** - общее время работы с базой данных;

queries – время и количество запросов к базе данных;

fetch – время и количество записей, извлечённых из результатов запросов к базе данных;

PHP total – общее время работы PHP-кода;

peak mem usage – максимальный объём памяти, выделенный скрипту ([memory\\_get\\_peak\\_usage\(\)](#)).

б) Инструкция `AMI::getSingleton('db')->displayQueries(true);` ([AMI\\_DB::displayQueries\(\)](#)) включает выдачу запросов к базе данных в отладочную информацию:

**»» Warning: debug mode is enabled, 2010-11-26 18:21:31 GMT: 2010-11-26 12:21**

Delay: 0.00332 sec, total: 0.00567 sec, total fetch count: 4

DESCRIBE my\_module1\_table\_item

Delay: 0.00340 sec, total: 0.00908 sec, total fetch count: 14

SELECT i.id,i.public,i.lang,i.header,i.announce,i.body,i.date\_created,i.date\_modified FROM my\_module1\_table\_item i

Delay: 0.00031 sec, total: 0.00938 sec, total fetch count: 17

SHOW CREATE TABLE `my\_module2\_table\_cat`

Delay: 0.00456 sec, total: 0.01394 sec, total fetch count: 20

DESCRIBE my\_module2\_table\_item

Delay: 0.00376 sec, total: 0.01771 sec, total fetch count: 31

SELECT cat.id cat\_id,cat.header cat\_header,cat.lang

cat\_lang,i.id,i.public,i.lang,i.header,i.announce,i.body,i.date\_created,i.date\_modified,cat.id cat\_id,cat.header cat\_header,cat.lang cat\_lang FROM my\_module2\_table\_item i INNER JOIN my\_module2\_table\_cat cat ON cat.id=i.id\_cat

Delay – время исполнения запроса;

total – общее время исполнения запросов;

total fetch count – время и количество записей, извлечённых из результатов запросов к базе данных.

в) Инструкция `d::b('label');` ([d::b\(\)](#)) позволяет добавлять точки профилирования:

**»» Warning: debug mode is enabled, 2010-11-26 18:47:59 GMT: 2010-11-26 12:47**

Script total: 0.049011 sec [DB total: 0.018188 sec / queries: 0.017823 sec (6 times) / fetch: 0.000365 sec (31 times)] [PHP total: 0.030823 sec / peak mem usage: 441 752 bytes]

[at start]: 0.01915 sec (total: 0.019) , mem diff: 259 736 bytes (peak: 273 432 bytes), models.php: 64

[at end]: 0.02959 sec (total: 0.049) , mem diff: 163 208 bytes (peak: 438 640 bytes), models.php: 117

0.02959 sec – время с предыдущего вызова [d::b\(\)](#) или со старта скрипта;

total – время со старта скрипта;

mem diff – разница объёма выделенной памяти с предыдущего вызова [d::b\(\)](#) или со старта скрипта ([memory\\_get\\_usage\(\)](#));

peak – максимальный объём памяти, выделенный скрипту ([memory\\_get\\_peak\\_usage\(\)](#)) в момент вызова [d::b\(\)](#);

models.php: 117 – имя файла и строка, из которой был вызван [d::b\(\)](#).

### 11.3 Средства отладки шаблонов

В Amiro.CMS также встроены средства для глубокой отладки шаблонов. Подробная инструкция по их использованию доступна в [документации по интеграции дизайна](#).

