

Создание
компонента для
Joomla

Введение

Структура программного обеспечения – основа, которая может использоваться разработчиком. Framework (структура) Joomla1.5 дает огромные возможности разработчикам.

В этой части мы создадим простой компонент который будет выводить Hello World. В следующих частях эта простая структура будет усложняться, что бы показать полные возможности MVC (Model-View-Controller) в Joomla!

Коротко распишем части компонента MVC:

Модель

Модель – это часть компонента, которая оперирует данными приложения. В нашем случае модель будет содержать методы добавления, удаления, и обновления информации. Таким образом если программа сохраняла данные в файлы, а нам нужно сохранять их в БД, то для этого нужно будет изменить только модель, не трогая остальной код.

Вид

Вид – это часть компонента, которая берет данные из модели и выводит их нам в пользовательский интерфейс. Для простых веб-приложений, Вид генерирует html страницу, с входными данными. Т.е. вид берет данные из модели, и вносит их в шаблон, который в конечном итоге и видит пользователь. Вид не изменяет данные, он только отображает их из модели!

Контроллер

Контроллер отвечает за действия пользователя. В нашем случае, действие пользователя – это запрос страницы. Контроллер в зависимости от запроса, будет вызывать модель, для управления данными, и управлять переходом Модели на Вид. Таким образом контроллер не отображает данные, а лишь вызывает модель, которая изменяет данные, а затем передает модель в вид, который отображается на экране.

Создание компонента

В зависимости от сложности компонентов есть дополнительные директории с файлами для Модели, Представления, и Контроллера.

Каждый компонент находится в своей директории. Joomla обрабатывает компонент в следующей последовательности:

Joomla! интерпретирует выполненные значения в URL: `/index.php?option=com_hello`.

Ищет составляющую таблицу `com_hello` компонент.

Затем ищет директорию `com_hello`.

В этой директории ищет входящий файл `hello.php`.

И исполняет этот файл.

Для нашего основного компонента нам потребуется только пять файлов:

`hello.php` – это входная точка в компонент, похожая на `index.php` для сайтов.

`controller.php` – этот файл содержит базовый контроллер

`views/hello/view.html.php` – этот файл Вида. Он получает необходимые данные и передает их в шаблон.

`views/hello/tmpl/default.php` – сам шаблон.

`hello.xml` – это XML файл, который дает указания Joomla как, куда и что устанавливать.

Создание точки входа (`hello.php`)

Joomla! всегда обрабатывает ссылку в корневом файле `index.php` для страниц Front End (сайт) или `administrator/index.php` для страниц Back End (админ). Функция обработки URL загрузит требуемый компонент, основанный на значении 'option' в URL (метод GET) или переданных данных методом POST.

Для нашего компонента, URL выглядит так:

index.php? option=com_hello&view=hello

Эта ссылка запустит выполнение файла, являющегося точкой входа в наш компонент: components/com_hello/hello.php.

Код для этого файла довольно типичен для всех компонентов (hello.php):

```
<?php

// Защита от прямого обращения к скрипту
defined( '_JEXEC' ) or die( 'Restricted access' );

// Подключение файла контроллера
require_once( JPATH_COMPONENT.DS.'controller.php' );

// По необходимости подключаем дополнительный контроллер.
if($controller = JRequest::getWord('controller')) {

    $path = JPATH_COMPONENT.DS.'controllers'.DS.
    $controller.'.php';

    if (file_exists($path)) {

        require_once $path;

    } else {

        $controller = '';

    }

}

// Создание класса нашего компонента
$classname    = 'HelloController'.$controller;
$controller   = new $classname( );

// Выполняем задачу
$controller->execute( JRequest::getVar( 'task' ) );

// Переадресация
$controller->redirect();
```

Рассмотрим код более подробно:

```
defined('_JEXEC') or die('Restricted access');
```

Первая строка – проверка безопасности, проверяется относится ли файл к Joomla! или ее директориям.

```
require_once (JPATH_COMPONENT_DS.'controller.php');
```

JPATH_COMPONENT – абсолютный путь к текущему компоненту, в наше случае components/com_hello.

Для Front End используется JPATH_COMPONENT_SITE

Для Back End используется JPATH_COMPONENT_ADMINISTRATOR

DS – автоматический выбором слеша (разделителя директорий) 'W' или '/', в зависимости от ОС. Таким образом разработчик не должен волноваться о том на какой системе установлена Joomla.

После загрузки основного контроллера проверяется необходимость загрузки дополнительного контроллера. В данном случае у нас только основной контроллер, но мы оставим эту часть для будущего использования.

```
$classname = 'HelloController';
```

```
$controller = new $classname( );
```

Метод JRequest::getVar находит переменную в URL или в POST данных.

Например, наш URL может выглядеть так index.php?option=com_hello&controller=controller_name, тогда мы можем получить имя нужного нам контроллера, используя:

```
echo JRequest::getVar("controller");
```

Сей час мы используем наш основной контроллер-класс HelloController в com_hello/controller.php, и, если необходимо, добавляем контроллер, например: HelloControllerController1 из com_hello/controllers/controller1.php.

Если у Вас будет один контроллер, как это часто требуется в программах пользовательского интерфейса, то можно использовать следующие операторы:

```
$controller=newHelloController();
```

```
$controller->execute( JRequest::getVar('task'));
```

После того, как контроллер создан, мы инструктируем контроллер, чтобы выполнить задачу, как определено в URL: index.php?option=com_hello&task=Задача. Если нет определение задачи, то по умолчанию будет задача "display". Когда используется задача 'display', переменная "display" укажет то что выводить. Пример стандартных задач – сохранить (save), редактировать (edit), создать (new) и т.л.

```
$controller->redirect();
```

Контроллер может принять решение о переадресации страницы, обычно это бывает после задачи “сохранить”.

Основная точка входа (hello.php) по существу передает контроль на контроллер, который обрабатывает задачу, которая была определена в запросе.

Примечание: Как Вы можете видеть в коде нет закрывающего php тега: ?>. Причина в том, что бы не было никакого нежелательного свободного места в коде вывода. Эта практика началась с Joomla 1.5 и используется во всех файлах содержащих только чистый php.

Создание Контроллера (controller.php)

Наш компонент имеет только одну задачу – вывести надпись “Hello”. Таким образом, контроллер будет очень простой. Манипуляций с данными не требуется. Все, что нужно сделать – это загрузить Вид. В нашем контроллере будет только один метод display(). Большая часть необходимых функций уже встроена в класс JController, так что все, что нам нужно сделать, это вызвать метод JController::display() .

Код простейшего основного контроллера (controller.php):

```

<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport('joomla.application.component.controller');
class HelloController extends JController
{
/**
* метод вывода на экран
*
* @access public
*/
function display()
{
parent::display();
}
}

```

Конструктор JController будет всегда регистрировать display() задачу так как у нас она одна и не определена (используется registerDefaultTask() метод), устанавливается default task метод (задача по умолчанию).

display () – это первый по приоритету метод, он вызывает родительский конструктор.

JController :: display () метод определит название layout и layout запроса и загрузки, которые определяют и устанавливают выбор ветки в дереве задач. Когда Вы создаете пункт меню для вашего компонента, менеджер меню позволит администратору выбирать задачу, с которой начинать выполнение компонента. К примеру, выбор при создании пункта меню таких задач: вывод списка автомобилей , вывод списка событий , вывод данных по одному автомобилю, вывод отдельного случая. Layout – путь, которым задается вывод определенной задачи.

Например – Вы создаете вид вывода категории списком или блогом.

Создание Представления (Вида) (/views/hello/view.html.php)

Задача очень проста: Извлекаем данные, которые будут отображаться, и передаем их в шаблон. Данные передаются в шаблон, с помощью метода `JView::assignRef()` класса `JView`.

Код вида будет (view.html.php):

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport( 'joomla.application.component.view' );
class HelloViewHello extends JView
{
function display($tpl = null)
{
    $greeting = "Hello World!";
    $this->assignRef( 'greeting', $greeting );
    parent::display($tpl);
}
}
```

Создание шаблона(/views/hello/tmpl/default.php)

Наш шаблон очень прост, мы только отображаем приветствие, которое передавали в view (default.php):

```
<?php
defined( '_JEXEC' ) or die( 'Restricted access' ); ?>
<h1><?php echo $this->greeting; ?></h1>
```

Создание установочного файла (hello.xml)

Можно установить компонент вручную, копируя файлы по FTP протоколу создавая необходимые папки и таблицы в базе данных, но лучшим вариантом является использования установочного файла для пакетной загрузки файлов и установки компонента.

Установочный XML файл может содержать разнообразную информацию и инструкции по установке:

Детали о компоненте и об авторе компонента;

Список файлов, которые должны быть скопированы;

Внешний PHP файл, который исполняет дополнительную установку и деинсталлирует операции;

Внешние SQL файлы, которые содержит запросы базы данных, отдельно для установки и деинсталляции.

Пример данного установочного XML файла:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE install SYSTEM
"http://dev.joomla.org/xml/1.5/component-install.dtd">
<install type="component" version="1.5.0">
  <name>Hello</name>
  <!-- Далее идут необязательные элементы-->
  <creationDate>20.02.2009</creationDate>
  <author>Имя автора</author>
  <authorEmail></authorEmail>
  <authorUrl>http://www.example.org</authorUrl>
  <copyright>GNU/GPL</copyright>
  <license>Информация о лицензии</license>
  <!-- Номер версии сохраняется как строковое значение в
таблице компонентов -->
```

```
<version>Версия компонента</version>
```

```
<!-- Описание необязательно, и по-умолчанию равно имени компонента -->
```

```
<description>Описание компонента ...</description>
```

```
<!--Копирование файлов во фронтенд -->
```

```
<files folder="site">
```

```
  <filename>index.html</filename>
```

```
  <filename>hello.php</filename>
```

```
  <filename>controller.php</filename>
```

```
  <filename>views/index.html</filename>
```

```
  <filename>views/hello/index.html</filename>
```

```
  <filename>views/hello/view.html.php</filename>
```

```
  <filename>views/hello/tmpl/index.html</filename>
```

```
  <filename>views/hello/tmpl/default.php</filename>
```

```
</files>
```

```
<administration>
```

```
  <!--меню в админке -->
```

```
  <menu>Hello World!</menu>
```

```
  <!-- копирование файлов в админку -->
```

```
  <files folder="admin">
```

```
    <filename>index.html</filename>
```

```
    <filename>admin.hello.php</filename>
```

```
  </files>
```

```
</administration>
```

```
</install>
```

Как видно из этого кода у нас должно быть две директории: site и admin. В директорию site копируем все созданные файлы, кроме hello.xml. В директории admin создаем пустой файл admin.hello.php

Также есть файлы, которые будут скопированы, это – index.html.

index.html помещен в каждый каталог, чтобы препятствовать пользователям получать листинг каталога.

Эти файлы содержат одну единственную строку:

```
<html><body bgcolor="#FFFFFF"></body></html>
```

С помощью этих файлов будет отображаться пустая страница.

В итоге весь листинг компонента будет таким:

Как посмотреть на свое творение?

Можно в браузере набрать `http://ДОМЕН/index.php?option=com_hello`. Так же можно создать пункт меню на компонент, для этого зайди в главное меню, дальше создай в нем новый пункт Hello Word! (разметка будет default, т.к. она одна), на сайте перейди на этот пункт откроется компонент com_hello.

Использование Модели

В первом уроке был создан простой компонент view-controller для CMS Joomla 1.5. Приветствие (Hello) было жестко прописано в Представлении. Это не соответствует MVC, так как Представление предназначено для того, чтобы отображать данные, а не содержать их.

В этой части расширим возможности и гибкость компонента "Hello" добавив к компоненту Модель (Model).

Создание модели (/models/hello.php)

Модель – часть компонента, которая предоставляет данные для Представления по его запросу посланному через Контроллер. Такой метод часто освобождает от рутинной работы и от хаоса в коде, предоставляет возможность управлять данными удобным способом в дополнение к коду, который посылает запрос данных из Модели.

В Joomla 1.5, Модель будет содержать классы функций : добавить, удалить и модернизировать информацию в таблицах базы данных. Также содержать методы восстановления списка таблиц базы данных.

Наименование моделей строятся следующим образом: название компонента, model, название модели.

В нашем случае это HelloModelHello.

В общем основная структура доступа к данным должна быть кратко описана в модели.

Таким способом, если обработку запроса данных необходимо изменить, Модель – единственный элемент, в который вносятся изменения , не затрагивая код Представления (View) или Контроллера (Controller).

При переносе точки запроса данных в общем алгоритме компонента, вносятся изменение только в код Представления.

В этой части, будем моделировать событие компонента "Hello", которое генерирует приветствие. Таким образом в компоненте будет один запрос к модели getGreeting (), который возвратит строку "Hello, World!".

Образец кода для класса Модели:

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );
jimport( 'joomla.application.component.model' );
class HelloModelHello extends JModel
{
    /**
     * Gets the greeting
```

```

    * @return string The greeting to be displayed to the
user
*/
function getGreeting()
{
    return 'Hello, World!';
}
}
jimport( 'joomla.application.component.model' );

```

Обратите внимание на эту строку. Функция `jimport` используется, чтобы загрузить файлы из структуры Joomla, которые требуются для нашего компонента. В данном случае загрузится файл `/libraries/joomla/application/component/model.php`. Точки используются как разделители директорий. Все файлы загружаются относительно директории библиотек – `libraries`.

В данном случае подключается файл из библиотеки – `model.php`. Это специальный файл, который описывает класс `JModel`, он нам необходим, потому что наша модель расширяет этот класс.

Теперь, когда мы создали модель, мы должны изменить Представление так, чтобы оно использовала данные от модели.

Использование модели

Структура Joomla! (framework) устроена таким образом, что контроллер автоматически загрузит модель в Представление, если Модель имеет то же название что и Представление. Так как у нас названия совпадают, то можно использовать метод `JView::getModel()`.

В предыдущей части Представление было таким:

```
$greeting = "Hello World!";
```

Теперь, с использованием Модели, оно будет таким:

```
$model =& $this->getModel();  
$greeting = $model->getGreeting();
```

Конечный код Представления (/views/hello/view.html.php) будет:

```
<?php  
  
// No direct access  
  
defined( '_JEXEC' ) or die( 'Restricted access' );  
jimport( 'joomla.application.component.view' );  
  
/**  
 * HTML View class for the HelloWorld Component  
 *  
 * @package      HelloWorld  
 */  
  
class HelloViewHello extends JView  
{  
  
    function display($tpl = null)  
    {  
  
        $model =& $this->getModel();  
        $greeting = $model->getGreeting();  
        $this->assignRef( 'greeting', $greeting );  
        parent::display($tpl);  
    }  
}
```

Добавление для файла hello.xml

Для завершения работы над данной версией компонента, необходима в секцию Site (Front End) добавить файлы Модели:

```
<filename>models/hello.php</filename>
```

Новый файл hello.xml будет иметь следующий вид:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<!DOCTYPE install SYSTEM
```

```
"http://dev.joomla.org/xml/1.5/component-install.dtd">
```

```
<install type="component" version="1.5.0">
```

```
<name>Hello</name>
```

```
<creationDate>November</creationDate>
```

```
<author>Nobody</author>
```

```
<authorEmail>nobody@example.org</authorEmail>
```

```
<authorUrl>http://www.example.org</authorUrl>
```

```
<copyright>Copyright Info</copyright>
```

```
<license>License Info</license>
```

```
<version>Component Version String</version>
```

```
<description>description of the component</description>
```

```
<!-- Site Main File Copy Section -->
```

```
<files folder="site">
```

```
<filename>models/hello.php</filename>
```

```
<filename>index.html</filename>
```

```
<filename>hello.php</filename>
```

```
<filename>controller.php</filename>
```

```
<filename>views/index.html</filename>
```

```
<filename>views/hello/index.html</filename>
```

```
<filename>views/hello/view.html.php</filename>
```

```
<filename>views/hello/tmpl/index.html</filename>
```

```
<filename>views/hello/tmpl/default.php</filename>
```

```
</files>
```

```
<administration>
<!-- Administration Menu Section -->
<menu>Hello World!</menu>
<!-- Administration Main File Copy Section -->
<files folder="admin">
<!-- Site Main File Copy Section -->
<filename>index.html</filename>
<filename>admin.hello.php</filename>
</files>
</administration>
</install>
```

Заключение

Мы теперь имеем простой MVC компонент. Каждый элемент очень прост в данный момент, но компонент уже обладает более большей гибкостью и возможностью.

Использование БД

В первых двух частях мы научились строить простой model-view-controller компонент. В итоге в компоненте есть одно Представление (View), которое получает данные от Модели, созданной в 2-ом уроке. Сей час, будем работать с моделью получая данные с таблицы базы данных.

Будет продемонстрировано, как использовать класс JDatabase для работы с базой данных.

Получение Данных

Наша модель в настоящее время имеет один метод: `getGreeting()`. Этот метод очень прост – все, что эта функция делает – возвращение жестко-закодированное приветствие.

Для работы с базой данных, загрузим приветствие в таблицу базы данных.

При установке компонента необходимо создание таблицы с записью текстового поля с приветствием. Как создать SQL файл и какие строки добавить в hello.xml, рассмотрим позже.

Для начала заменим код в Моделе для получения приветствия с таблицы базы данных.

В первую очередь нужно подключиться к базе данных. В Joomla 1.5 все параметры для подключения уже есть и для того чтобы создать свое используем следующее:

```
$db =& JFactory::getDBO();
```

JFactory – статический класс, который используется, чтобы подключиться к многим объектам системы. Подробней информацию об этом классе можно посмотреть в документации API.

Для подключения к Базе данных используется метод getDBO.

Два шага для получение приветствия:

Сохраняем запрос для получения объекта базы данных;

Загружаем результат;

Для этого вносим изменение для Модели в методе getGreeting() (/models/hello.php):

```
function getGreeting()
{
    $db =& JFactory::getDBO();
    $query = 'SELECT greeting FROM #__hello';
    $db->setQuery( $query );
    $greeting = $db->loadResult();
    return $greeting;
}
```

\$db->loadResult() метод выполнит запрос к базе данных, и вернет полученный объект. Подробнее об этом методе можно посмотреть в [JDatabase API reference](#), а так же изучить остальные методы JDatabase класса.

Создаем инсталляционный SQL файл (/admin/install.sql)

Joomla 1.5 инсталлятор имеет встроенную поддержку выполнения запросов SQL в процессе установки компонента. Эти запросы должны быть сохранены в стандартном install.sql файле.

Для инсталляционного файла SQL используем три запроса:

Удаление таблицы на случай, если с таким именем уже существует.

Создание таблицы и текстового поля для хранения строки приветствия.

Загрузка строки приветствия в поле таблицы.

Ниже все три запроса для инсталляционного файла:

```
DROP TABLE IF EXISTS `#__hello`;  
  
CREATE TABLE `#__hello` (  
    `id` int(11) NOT NULL AUTO_INCREMENT,  
    `greeting` varchar(25) NOT NULL,  
    PRIMARY KEY (`id`)  
)  
ENGINE=MyISAM AUTO_INCREMENT=0 DEFAULT CHARSET=utf8;  
  
INSERT INTO `#__hello` (`greeting`) VALUES ('Hello,  
World!'),  
('Bonjour, Monde!'),  
('Ciao, Mondo!');
```

Joomla сама заменит "#__" на префикс таблицы текущей базы данных MySQL. в первом запросе уничтожается таблица #__hello, это необходимо для того, чтобы не было накладок при повторных установках одного и того же компонента.

По второму запросу создается два поля в таблице. 1. `ID` – которое является ключом, который гарантирует уникальность записи. 2. `greeting` – строка длиной 25 символов, в которой будет храниться приветствие.

Сохраняем эти запросы в файле `install.sql` и переносим его в дистрибутив компонента по пути – `/admin/install.sql`.

Создаем деинсталляционный SQL файл (`/admin/uninstall.sql`)

При деинсталляции компонента необходимо удалять таблицы, которые были созданы установкой данного, но при этом нужно всегда учитывать, то что пользователь может случайно удалить компонент. Для того чтобы он таким образом не удалил свои данные в таблицах базы данных, необходимо требовать подтверждение такого действия:

```
DROP TABLE IF EXISTS `#__hello`;
```

Сохраняем эти запросы в файле `uninstall.sql` и переносим его в дистрибутив компонента по пути – `/admin/uninstall.sql`.

Заносим дополнение в инсталляционный файл (`hello.xml`)

Прежде чем указывать какие файлы использовать при установке и деинсталляции для запуска SQL запросов, нужно указать куда копировать эти файлы. Оба файла должны находиться в корне папки административной части компонента. Далее, указываем инсталлятору какие файлы использовать для SQL запроса при установке и деинсталляции компонента.

Новый код для инсталляционного XML файла:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<install type="component" version="1.5.0">
```

```
<name>Hello</name>
```

```
<!-- The following elements are optional and free of  
formatting constraints -->
```

```
<creationDate>2007-02-22</creationDate>
```

```
<author>John Doe</author>
```

```
<authorEmail>john.doe@example.org</authorEmail>
```

```
<authorUrl>http://www.example.org</authorUrl>
```

```
<copyright>Copyright Info</copyright>
```

```
<license>License Info</license>
```

```
<!-- The version string is recorded in the components  
table -->
```

```
<version>3.01</version>
```

```
<!-- The description is optional and defaults to the  
name -->
```

```
<description>Description of the component</description>
```

```
<!-- Site Main File Copy Section -->
```

```
<!-- Note the folder attribute: This attribute describes  
the folder
```

```
to copy FROM in the package to install therefore  
files copied
```

```
in this section are copied from /site/ in the  
package -->
```

```
<files folder="site">
```

```
<filename>controller.php</filename>
```

```
<filename>hello.php</filename>
```

```
<filename>index.html</filename>
```

```
<filename>models/hello.php</filename>
```

```
<filename>models/index.html</filename>
```

```
<filename>views/index.html</filename>
```

```
<filename>views/hello/index.html</filename>
```

```
<filename>views/hello/view.html.php</filename>
```

```
<filename>views/hello/tmpl/default.php</filename>
```

```
<filename>views/hello/tmpl/index.html</filename>
```

```

</files>

<install>

  <sql>

    <file charset="utf8" driver="mysql">install.sql</file>

  </sql>

</install>

<uninstall>

  <sql>

    <file charset="utf8"
driver="mysql">uninstall.sql</file>

  </sql>

</uninstall>

<administration>

  <!-- Administration Menu Section -->

  <menu>Hello World!</menu>

  <!-- Administration Main File Copy Section -->

  <files folder="admin">

    <filename>hello.php</filename>

    <filename>index.html</filename>

    <filename>install.sql</filename>

    <filename>uninstall.sql</filename>

  </files>

</administration>

</install>

```

При установке и деинсталляции может быть два значения у атрибута charset, первый - "utf8" и если версия MySQL сервера не поддерживает кодировку utf8 указывается атрибут "non-utf8"

Атрибут `driver` пока может иметь только одно значение – `"mysql"`. В дальнейшем планируется расширить возможности Joomla 1.5 для работы с разными Базами данных.

Заключение

Теперь компонент "Hello" может работать не только с MVC framework классами, но и с классами запросов JDatabase. Вы уже можете создавать компоненты работающее по технологии MVC с использованием таблиц базы данных, указывая их параметры инсталлятору.

Интерфейс Администратора

В предыдущей части у компонента появилась возможность получать информацию из базы данных и выводить ее, но он все еще не умеет изменять или добавлять новые записи в таблицу базы данных.

В этой части добавим к компоненту "Hello" в раздел Администратора интерфейс для работы с записями в таблице базы данных.

Создание основной структуры (/admin/hello.php)

Основная среда панели администратора очень похожа на часть сайта. Основной точкой входа в администраторскую часть компонента является `hello.php`. Этот файл идентичен файлу `hello.php`, который используется на сайте, кроме того, что имя загружаемого контроллера, изменено на `HelloController`. Контроллер по умолчанию также называется `controller.php`, и является идентичным контроллеру по умолчанию в части сайта, с тем отличием, что контроллер называется `HelloController` вместо `HelloController`. Эта разница означает, что контроллер `JController` по умолчанию будет загружать список наших приветствий .

Код для файла `hello.php`:

```
<?php
```

```
// No direct access
```

```

defined( '_JEXEC' ) or die( 'Restricted access' );
// Require the base controller
require_once( JPATH_COMPONENT.DS.'controller.php' );
// Require specific controller if requested
if($controller = JRequest::getWord('controller')) {
    $path = JPATH_COMPONENT.DS.'controllers'.DS.
$controller.'.php';

    if (file_exists($path)) {
        require_once $path;
    } else {
        $controller = '';
    }
}

// Create the controller
$classname    = 'HelloController'.$controller;
$controller   = new $classname( );

// Perform the Request task
$controller->execute( JRequest::getVar( 'task' ) );

// Redirect if set by the controller
$controller->redirect();

```

Представлением и моделью, с которых мы начнем, будут, соответственно, hellos view и hellos model. Начнем с модели.

Модель Hellos (/admin/models/hellos.php)

Модель Hellos будет очень простой . Единственным действием, нужным нам сей час, является возможность получения списка приветствий из базы данных. Это действие будет выполняться в методе getData().

Класс JModel имеет встроенный защищенный метод `_getList()`. Этот метод может использоваться для упрощения задачи получения списка записей из базы данных. Мы должны просто выполнить запрос, и он возвратит список записей .

Позже может возникнуть необходимость использовать запрос в другом методе. Таким образом, создадим защищенный метод `_buildQuery()`, который возвратит запрос, передаваемый методу `_getList()`. Это облегчает изменение запроса, поскольку он вызывается из одного и того же места.

Таким образом, в нашем классе нужны 2 метода: `getData()` и `_buildQuery()`.

`_buildQuery()` просто возвращает запрос. Это выглядит примерно так:

```
function _buildQuery()  
{  
    $query = ' SELECT * '  
        . ' FROM #__hello '  
    ;  
    return $query;  
}
```

`getData()` получает запрос и извлекает записи из базы данных. Может случиться, что нам потребуется дважды получить этот список при одной загрузке страницы. Будет расточительством получать этот список дважды. Таким образом, метод должен сохранять полученные данные в защищенном свой стве, чтобы на последующий запрос он мог просто возвращать полученные ранее данные. Это свой ство будет называться `_data`.

Ниже представлен метод `getData()`:

```
/**  
 * Получение данных
```

```

* @Возвращает Массив объектов, содержащие данные из БД
*/
function getData()
{
    // Загрузка данных, если они еще не были загружены
    if (empty( $this->_data ))
    {
        $query = $this->_buildQuery();
        $this->_data = $this->_getList( $query );
    }

    return $this->_data;
}

```

Полностью модель Hellos будет выглядеть так:

```

<?php
// Check to ensure this file is included in Joomla!
defined( '_JEXEC' ) or die();
jimport( 'joomla.application.component.model' );
/**
 * Hello Model
 *
 * @package Joomla.Tutorials
 * @subpackage Components
 */
class HellosModelHellos extends JModel
{
    /**

```

```

    * Hellos data array
    *
    * @var array
    */
var $_data;

/**
 * Returns the query
 * @return string The query to be used to retrieve
the rows from the database
 */

function _buildQuery()
{
    $query = ' SELECT * '
        . ' FROM #__hello '
        ;
    return $query;
}

/**
 * Retrieves the hello data
 * @return array Array of objects containing the data
from the database
 */

function getData()
{
    // Lets load the data if it doesn't already exist
    if (empty( $this->_data ))
    {

```

```

        $query = $this->_buildQuery();

        $this->_data = $this->_getList( $query );
    }

    return $this->_data;
}
}

```

Файл сохраним как /admin/models/hellos.php

Представление (Вид) Hellos (/admin/views/hellos/view.html.php)

Итак, у нас есть данные которые берет модель. Теперь нужно отобразить эти данные с помощью Представления. Представление в Административной части почти точно такое же как и на сайте (front-end).

В Представлении будем использовать метод get() класса JView. У Представления будет три строчки: первая берет данные из модели, вторая помещает данные в шаблон, третья отображает данные. Таким образом, мы имеем:

```

<?php

// Check to ensure this file is included in Joomla!
defined('_JEXEC') or die();

jimport( 'joomla.application.component.view' );

/**
 * Hellos View
 *
 * @package Joomla.Tutorials
 * @subpackage Components
 */

class HellosViewHellos extends JView
{

```

```

/**
 * Hellos view display method
 * @return void
 **/

function display($tpl = null)
{
    JToolBarHelper::title( JText::_( 'Hello
Manager' ), 'generic.png' );

    JToolBarHelper::deleteList();
    JToolBarHelper::editListX();
    JToolBarHelper::addNewX();

    // Get data from the model
    $items =& $this->get( 'Data' );
    $this->assignRef( 'items', $items );
    parent::display($tpl);
}
}

```

Сохранем файл как /admin/views/hellos/view.html.php.

Шаблон Hellos (/admin/views/hellos/tmpl/default.php)

Как писалось ранее шаблон берет данные от Представления и производит их вывод. Отообразим данные через цикл в виде простой таблицы:

```

<?php defined( '_JEXEC' ) or die( 'Restricted access' ); ?>
<form action="index.php" method="post" name="adminForm">
<div id="editcell">
    <table class="adminlist">

```

```
<thead>
  <tr>
    <th width="5">
      <?php echo JText::_ ( 'ID' ); ?>
    </th>
    <th>
      <?php echo JText::_ ( 'Greeting' ); ?>
    </th>
  </tr>
</thead>
<?php
$k = 0;
for ($i=0, $n=count( $this->items ); $i < $n; $i++)
{
  $row =& $this->items[$i];
  ?>
  <tr class="<?php echo "row$k"; ?>">
    <td>
      <?php echo $row->id; ?>
    </td>
    <td>
      <?php echo $row->greeting; ?>
    </td>
  </tr>
  <?php
  $k = 1 - $k;
}
```

```
    ?>
    </table>
</div>
<input type="hidden" name="option" value="com_hello" />
<input type="hidden" name="task" value="" />
<input type="hidden" name="checked" value="0" />
<input type="hidden" name="controller" value="hello" />
</form>
```

Этот шаблон сохранен как `views/hellos/tmpl/default.php`.

Обратите внимание: вывод заключен в форму. Сей час это не является необходимым, но скоро понадобится.

Мы завершили основную часть первого представления, добавив путь файлов в администраторский раздел (`admin`) нашего компонента.:

`hello.php`

`controller.php`

`models/hellos.php`

`views/hellos/view.html.php`

`views/hellos/tmpl/default.php`

Вы можете добавить эти файлы в XML-файл инсталляции и посмотреть, что получится!

Добавление функциональности

Итак, пока наш администраторский раздел не очень полезен. Пока он не делает ничего, кроме отображений содержимого базы данных.

Для добавления полезных функций необходимо добавить несколько кнопок и ссылок.

Панель инструментов

Возможно, вы обратили внимание на панель инструментов, отображаемую вверху панелей администратора компонент Joomla. Нашему компоненту она также необходима. Joomla облегчает ее создание. Добавим кнопки Удалить записи, Изменить записи, и Создать новые записи. Также добавим заголовок, который будет отображаться на нашей панели инструментов.

Это можно сделать, добавив немного кода в Представление. Чтобы добавить кнопки, используем статические методы из класса JToolBarHelper. Код выглядит так:

```
JToolBarHelper::title( JText::_( 'Hello Manager' ),  
'generic.png' );  
  
JToolBarHelper::deleteList();  
  
JToolBarHelper::editListX();  
  
JToolBarHelper::addNewX();
```

Эти три метода создают соответствующие кнопки. Метод deleteList() может принимать три параметра: первый параметр является строкой, спрашивающей пользователя о подтверждении удаления. Второй параметр является задачей, которая отправляется вместе с запросом (по умолчанию "remove"), а третий – текст, отображаемый под кнопкой.

Методы editListX() и addNewX() могут получать два дополнительных параметра. Первый – задача (по умолчанию – соответственно, edit и add), второй – текст, отображаемый под кнопкой.

Возможно, вы обратили внимание на использование метода JText::_ как в прошлом шаблоне, так и здесь. Это функция, значительно облегчающая перевод компонента. Метод JText::_ ищет текстовую строку в языковом файле компонента и возвращает переведенную строку. Если перевод не найден, функция возвращает

строку, переданную ей . Если компонент нужно перевести на другой язык, все, что нужно сделать – создать языковой файл, включающий строки и их перевод на требуемый язык.

Флажки и ссылки

Теперь у нас есть кнопки. Две из этих кнопок управляют существующими записями. Но как узнать, с какими именно записями необходимо работать? Пусть это определит пользователь. Для этого нам нужно добавить флажки в таблицу, чтобы пользователь мог выбрать необходимые записи. Это реализовано в нашем шаблоне.

Для добавления флажков нам необходимо добавить в таблицу дополнительный столбец. Мы добавим столбец между двумя имеющимися.

В заголовке столбца добавим флажок, который можно использовать для выбора или сброса всех флажков:

```
<th width="20">  
    <input type="checkbox" name="toggle" value=""  
onclick="checkAll();" />  
</th>
```

Функция Javascript checkAll встроена в основной пакет Joomla! Javascript, предоставляющий нужную нам функциональность.

Теперь необходимо добавить флажки в каждую строку. У класса JHTML есть метод JHTML::_(), который создаст для нас флажки. Добавим следующие строки в наш цикл:

```
$checked = JHTML::_( 'grid.id', $i, $row->id );
```

После линии:

```
$row =& $this->items[$i];
```

Затем добавим ячейку между двумя имеющимися:

```
<td>  
    <?php echo $checked; ?>
```

```
</td>
```

Необходимость выбирать флажок, перемещаться вверх страницы и нажимать кнопку (Edit) слишком обременительно для редактирования приветствия. Мы добавим ссылку, позволяющую перейти непосредственно к форме редактирования. Следующие строки добавим после вызова метода `JHTML::_()` для создания ссылки HTML:

```
$link = JRoute::_( 'index.php?option=com_hello&controller=hello&task=edit&cid[]=' . $row->id );
```

Добавляем ссылку в ячейку, отображающую название приветствия:

```
<td>
    <a href="/<?php echo $link; ?>"><?php echo $row->greeting; ?></a>
</td>
```

Обратите внимание, что ссылка указывает на контроллер `hello`. Этот контроллер обрабатывает данные наших приветствий .

Если вы помните, у нас были четыре скрытых поля внизу формы. Первое поле носило имя `option`. Вторым полем является поле `task`. Оно получает данные в случае нажатия одной из кнопок на панели инструментов. В случае удаления этого поля будет получена ошибка Javascript и кнопки не будут работать. Третье поле – `checked`. Оно хранит количество отмеченных флажков. Кнопки редактирования и удаления проверяют условие превышения этой величиной нуля, в противном случае не позволяя отправку данных формы. Четвертое поле – это поле контроллера, используемое для определения того, что данные, отправленные из этой формы, будут обработаны контроллером `hello`.

Вот полный код шаблона файла `default.php`:

```
<?php defined( '_JEXEC' ) or die( 'Restricted access' ); ?>
<form action="index.php" method="post" name="adminForm">
```

```

<div id="editcell">
    <table class="adminlist">
    <thead>
        <tr>
            <th width="5">
                <?php echo JText::_ ( 'ID' ); ?>
            </th>
            <th width="20">
                <input type="checkbox" name="toggle"
value="" onclick="checkAll(<?php echo count( $this-
>items ); ?>);" />
            </th>
            <th>
                <?php echo JText::_ ( 'Greeting' ); ?>
            </th>
        </tr>
    </thead>
    <?php
    $k = 0;
    for ($i=0, $n=count( $this->items ); $i < $n; $i++)
    {
        $row =& $this->items[$i];
        $checked      = JHTML::_ ( 'grid.id', $i, $row-
>id );
        $link = JRoute::_ ( 'index.php?
option=com_hello&controller=hello&task=edit&cid[]='. $row-
>id );
        ?>
        <tr class="<?php echo "row$k"; ?>">

```

```

        <td>
            <?php echo $row->id; ?>
        </td>
        <td>
            <?php echo $checked; ?>
        </td>
        <td>
            <a href="/<?php echo $link; ?>"><?php
echo $row->greeting; ?></a>
        </td>
    </tr>
    <?php
        $k = 1 - $k;
    }
    ?>
</table>
</div>
<input type="hidden" name="option" value="com_hello" />
<input type="hidden" name="task" value="" />
<input type="hidden" name="boxchecked" value="0" />
<input type="hidden" name="controller" value="hello" />
</form>

```

Теперь представление hellos закончено. Можно испытать компонент, чтобы увидеть результаты.

Опускаемся на землю: Выполнение реальной работы

Теперь, когда представление Hellos завершено, время обратить внимание на представление и модель Hello. Настоящая работа выполняется именно здесь.

Контроллер Hello

Единственной работой контроллера по умолчанию является отображение представлений .

Вы должны иметь возможность выполнять задачи, запускаемые из представления Hellos: добавлять, изменять и удалять.

Фактически, добавление и изменение являются одним и тем же заданием: они оба отображают пользователю форму, позволяющую редактировать приветствие. Единственная разница в том, что при создании отображается пустая форма, а при изменении – форма с данными. Поскольку они являются похожими, мы будем выполнять задачу добавления с помощью обработчика задачи изменения. Это указывается в нашем конструкторе:

```
function __construct()  
{  
    parent::__construct();  
    // Register Extra tasks  
    $this->registerTask( 'add' ,      'edit' );  
}
```

Первый параметр JController::registerTask является задачей , второй – метод ее выполнения.

Начнем с обработки задачи изменения. В этом случае работа контроллера проста. Все, что ему нужно – указать представление и макет для загрузки (в нашем случае представление hello и макет формы). Мы также укажем Joomla отключить главное меню во время изменения приветствия. Это предотвращает оставление открытых несохраненных записей :

```
function edit()
```

```
{  
  
    JRequest::setVar( 'view', 'hello' );  
  
    JRequest::setVar( 'layout', 'form' );  
  
    JRequest::setVar( 'hidemainmenu', 1 );  
  
    parent::display();  
  
}
```

Представление Hello

Представление Hello отображает форму, позволяющую пользователю редактировать приветствие. Метод `display` должен выполнять несколько простых операций :

1. получить данные из модели
2. создать панель инструментов
3. поместить данные в шаблон
4. вызвать метод `display()` для отрисовки шаблона

Это немного сложнее, так как одно представление выполняет как редактирование, так и добавление. Наша панель инструментов должна сообщать пользователю о выполняемой в данный момент операции – добавление или редактирование, то есть нужно определить выполняемую задачу.

Когда мы получаем запись для отображения из модели, мы можем использовать эти данные для определения текущей задачи. Если задачей является редактирование, значит, поле `id` записи было изменено. Если это новая задача, значит, его значение не будет установлено. Эта деталь может помочь определить, создается ли новая запись, или редактируется существующая.

Также добавим на панель инструментов две кнопки: `save` и `cancel`. Функциональность будет практически одинаковой, но в зависимости от текущей задачи отображаться будут разные кнопки. В случае новой записи будет отображаться кнопка `cancel`, а в случае изменения существующей – кнопка `close`.

Итак, метод display будет выглядеть так:

```
function display($tpl = null)
{
    //get the hello
    $hello      =& $this->get('Data');
    $isNew      = ($hello->id < 1);
    $text = $isNew ? JText::_( 'New' ) : JText::_( 'Edit'
);

    JToolBarHelper::title(  JText::_( 'Hello' ).':
<small><small>[ ' . $text.' ]</small></small>' );

    JToolBarHelper::save();

    if ($isNew) {
        JToolBarHelper::cancel();
    } else {
        // for existing items the button is renamed
`close`

        JToolBarHelper::cancel( 'cancel', 'Close' );
    }

    $this->assignRef('hello', $hello);

    parent::display($tpl);
}
```

Модель Hello

Для нашего представления необходимы данные. Это значит, что нужно создать соответствующую модель.

У нашей модели будут два свойства: `_id` и `_data`. `_id` будет хранить идентификатор приветствия, `data` – данные.

Начнем с конструктора, который получает `id` из запроса:

```
function __construct()
```

```

{
    parent::__construct();
    $array = JRequest::getVar('cid', 0, '', 'array');
    $this->setId((int)$array[0]);
}

```

Метод `JRequest::getVar()` используется для получения данных из запроса. Первым параметром является имя переменной формы. Вторым параметром – значение по умолчанию для присвоения в случае, если значение не найдено. Третьим параметром – это имя хэша для получения значения из `get`, `post`, и т.д., и последнее значение – тип данных, который следует установить для значения.

Конструктор получит первое значение из массива `cid` и присвоит его `id`.

Метод `setId()` может использоваться для установки `id`. Изменение `id`, на которое указывает наша модель, означает, что `points` указывает на неправильные данные. Следовательно, устанавливая значение `id`, мы очищаем свойство `data`:

```

function setId($id)
{
    // Устанавливаем id и удаляем данные
    $this->_id      = $id;
    $this->_data    = null;
}

```

Наконец, нам нужен метод для получения данных: `getData()`

`getData` проверит, установлено ли значение свойства `_data`. Если да, он просто возвратит его. В противном случае будут получены данные из базы данных:

```

function &getData()
{

```

```

// Загружаем данные
if (empty( $this->_data )) {
    $query = ' SELECT * FROM #__hello '.
        ' WHERE id = '.$this->_id;
    $this->_db->setQuery( $query );
    $this->_data = $this->_db->loadObject();
}

if (!$this->_data) {
    $this->_data = new stdClass();
    $this->_data->id = 0;
    $this->_data->greeting = null;
}

return $this->_data;
}

```

Форма

Наконец, все что нам осталось – создать форму для данных. Поскольку мы определили макет как форму, форма будет размещена в файле каталога tmpl представления hello под именем form.php:

```

<?php defined('_JEXEC') or die('Restricted access'); ?>
<form action="index.php" method="post" name="adminForm"
id="adminForm">
<div class="col100">
    <fieldset class="adminform">
        <legend><?php echo JText::_('Details'); ?>
    </legend>
    <table class="admintable">
    <tr>
        <td width="100" align="right" class="key">

```

```

        <label for="greeting">
            <?php echo JText::_ ( 'Greeting' ); ?
>:
        </label>
    </td>
    <td>
        <input class="text_area" type="text"
name="greeting" id="greeting" size="32" maxlength="250"
value="<?php echo $this->hello->greeting;?>" />
    </td>
</tr>
</table>
</fieldset>
</div>
<div class="clr"></div>
<input type="hidden" name="option" value="com_hello" />
<input type="hidden" name="id" value="<?php echo $this-
>hello->id; ?>" />
<input type="hidden" name="task" value="" />
<input type="hidden" name="controller" value="hello" />
</form>

```

Обратите внимание: в дополнение к полю ввода присутствует скрытое поле для id. Пользователь не должен изменять id, поэтому мы просто незаметно помещаем его в форму.

Реализация функций

Итак, наш контроллер выполняет только две задачи: создание и изменение. Однако у нас есть кнопки также для сохранения,

удаления записей , и отмены. Нужно написать соответствующий код для выполнения этих задач.

Сохранение записи

Следующим шагом логично будет реализовать сохранение записи. Это потребует использование выбора для обработки различных ситуаций , например, различия между созданием новой записи (запрос INSERT), и обновлением существующей записи (запрос UPDATE). Также существует несколько нюансов, связанных с получением данных из формы и помещения их в запрос.

Фреймворк Joomla! облегчает выполнение многих задач. Класс JTable упрощает управление записями в базе данных без необходимости заботиться о написании SQL-кода, лежащего в основе этих операций . Он также облегчает перенос данных из HTML-форм в базу данных.

Создание класса Table (/tables/hello.php)

Класс JTable является абстрактным классом, от которого можно получить производные классы для работы с конкретными таблицами. Для его использования нужно просто создать класс, расширяющий класс JTable, добавить поля вашей базы данных как свойства, и переопределить конструктор для указания имени таблицы и первичного ключа.

Вот как выглядит наш класс JTable:

```
<?php
// No direct access
defined( '_JEXEC' ) or die( 'Restricted access' );
/**
 * Hello Table class
 *
 * @package Joomla.Tutorials
 * @subpackage Components
```

```

*/
class TableHello extends JTable
{
    /**
     * Primary Key
     *
     * @var int
     */
    var $id = null;
    /**
     * @var string
     */
    var $greeting = null;
    /**
     * Constructor
     *
     * @param object Database connector object
     */
    function TableHello( &$db ) {
        parent::__construct('#__hello', 'id', $db);
    }
}

```

Как видите, здесь мы определили два поля: идентификатор и приветствие. Затем был определен конструктор, вызывающий конструктор родительского класса и передающий ему имя таблицы (#__hello), имя поля, являющегося первичным ключом (id), и объект конектора базы данных.

Этот файл следует назвать `hello.php` и поместить в каталог `tables` в администраторском разделе нашего компонента.

Реализация функций в модели

Теперь мы готовы добавить метод в модель для сохранения записи. Назовем этот метод `store`. Метод `store()` будет выполнять три вещи: помещать данные из формы в объект `TableHello`, проверять корректность сформированной записи и сохранять запись в базе данных.

Метод будет выглядеть так:

```
function store()
{
    $row =& $this->getTable();
    $data = JRequest::get( 'post' );
    // Bind the form fields to the hello table
    if (!$row->bind($data)) {
        $this->setError($this->_db->getErrorMsg());
        return false;
    }
    // Make sure the hello record is valid
    if (!$row->check()) {
        $this->setError($this->_db->getErrorMsg());
        return false;
    }
    // Store the web link table to the database
    if (!$row->store()) {
        $this->setError($this->_db->getErrorMsg());
        return false;
    }
}
```

```
        return true;
    }
}
```

Этот метод добавляется в модель Hello.

Метод получает один параметр, являющийся ассоциативным массивом данных, которые мы сохраняем в базу данных. Эти данные могут быть легко получены из запроса, как будет показано далее.

Первая строка получает ссылку на объект JTable. Если таблица названа правильно, мы можем не указывать это имя – класс JModel знает, где его искать. Как вы помните, мы назвали наш класс таблицы TableHello и поместили его в файл hello.php в каталоге tables. Если вы следовали этим рекомендациям, класс JModel создаст объект автоматически.

Вторая строка получает данные из формы. Класс JRequest делает эту операцию очень легкой. В данном случае мы получаем все переменные, переданные с помощью метода POST. Они возвращаются в виде ассоциативного массива.

Остальное просто – мы получаем, проверяем и сохраняем. Метод bind() копирует значения из массива в соответствующие свойства объекта таблицы. В данном случае он копирует значения идентификатора и приветствия в объект TableHello.

Метод check() выполняет проверку данных. В классе JTable() этот метод просто возвращает true. Пока он не представляет какого-либо значения, но в будущем он позволит проверять данные с помощью класса TableHello. Этот метод может быть переименован в классе TableHello методом, выполняющим необходимые проверки.

Метод store() будет помещать данные из объекта в базу данных. Если id равно нулю, будет создана новая запись (INSERT), в противном случае он обновит существующую запись (UPDATE).

Добавление задачи в контроллер

Теперь все готово для добавления задачи в контроллер. Поскольку задача называется save, мы должны назвать метод "save". Это просто:

```
function save()
{
    $model = $this->getModel('hello');
    if ($model->store()) {
        $msg = JText::_( 'Приветствие сохранено!' );
    } else {
        $msg = JText::_( 'Ошибка сохранения
приветствия' );
    }
    // Проверяем, возможно ли изменение таблицы....
    $link = 'index.php?option=com_hello';
    $this->setRedirect($link, $msg);
}
```

Все, что нам нужно – вызвать метод store() модели. Затем следует использовать метод setRedirect() для перенаправления к списку приветствий . Также мы задаем сообщение, которое будет отображено вверху страницы.

- Удаление записи
- Реализация функции в модели

В модели мы получаем список ID для удаления и вызываем класс JTable для их удаления:

```
function delete()
{
    $cids = JRequest::getVar( 'cid', array(0), 'post',
'array' );
    $row =& $this->getTable();
```

```

foreach($cids as $cid) {
    if (!$row->delete( $cid )) {
        $this->setError( $row->getErrorMsg() );
        return false;
    }
}

return true;
}

```

Мы вызываем метод `JRequest::getVar()` для получения данных из запроса, затем вызываем метод `delete()` для удаления каждой строки. Сохраняя ошибки в модели, мы обеспечиваем возможность получить их позже, если потребуется.

Выполнение задачи удаления в контроллере

Это очень похоже на метод `save()`, выполняющий сохранение:

```

function remove()
{
    $model = $this->getModel('hello');

    if (!$model->delete()) {
        $msg = JText::_( 'Error: One or More Greetings
Could not be Deleted' );
    } else {
        $msg = JText::_( 'Приветствие удалено' );
    }

    $this->setRedirect( 'index.php?option=com_hello',
$msg );
}

```

Отмена операции редактирования

Все, что нужно для прерывания операции редактирования – перенаправление на главное представление:

```
function cancel()  
{  
    $msg = JText::_( 'Операция прервана' );  
    $this->setRedirect( 'index.php?option=com_hello',  
$msg );  
}
```

Заключение

Мы реализовали простой механизм для нашего компонента. Теперь у нас есть возможность редактировать элементы, отображаемые в представлении. Мы продемонстрировали взаимодействие между моделями, представлениями и контроллерами. Также мы показали, как класс JTable можно расширить для предоставления простого доступа к таблицам в базе данных. Также можно увидеть использование класса JToolBarHelper для создания панелей кнопок в компоненте для предоставления стандартного вида для различных компонентов.