Разработчикам (API) - API для интеграции со сторонними приложениями.

www.amiro.ru

© 2011 Amiro.CMS

Оглавление

API для интеграции со сторонними приложениями

Набор классов User Source Application (AMI_UserSourceApp.php) совместно с точкой входа для сервисов (ami_service.php) позволяют как авторизовывать пользователей сторонних сервисов (например Twitter, Facebook и т.д.) на сайте под управлением Amiro.CMS, так и разрабатывать сервисы для авторизации пользователей в сторонних приложениях через сайт.

В стандартную поставку системы включены драйверы для авторизации пользователя на сайте с использованием учетных записей в социальных сервисах: Twitter, Facebook, VKontakte.

Инструкция по настройке встроенных драйверов авторизации.

Программный интерфейс

Чтобы создать и подключить новый драйвер для авторизации

- Создайте потомка класса **AMI_UserSourceAppDriver**

```
class Test_UserSourceAppDriver extends AMI_UserSourceAppDriver{

}
```

- Задайте свойству **driverId** нового класса уникальное значение (помните, что значения от 0 до 10000 системные и не могут быть заняты)

```
    /**
 * Driver id numeric (unique).
 *
 * @var void
 */
    public $driverId = 10001;
```

- Задайте свойству **driverName** новое уникальное название

```
/**
 * Driver name.
 *
 * @var void
 */
public $driverName = 'ami_test';
```

- Внесите свой драйвер в список ресурсов в ветку **'user_source_app/drivers/'**

```
  // User Source Applications
'user_source_app'                  => 'AMI_UserSourceApp',
'user_source_app/drivers/ami_twitter'   => 'Twitter_UserSourceAppDriver',
'user_source_app/drivers/ami_vkontakte' => 'VKontakte_UserSourceAppDriver',
'user_source_app/drivers/ami_facebook'  => 'Facebook_UserSourceAppDriver',
```

```
'user_source_app/drivers/ami_test'     => 'Test_UserSourceAppDriver',
  // } User Source Applications
```

- Если потребуется - создайте новый раздел в конфигурационном файле, и задайте новые настройки. Имя раздела должно соответствовать значению свойства **driverName**.

```
[ami_test]

enabled = yes
myoption = ‘test’;
```

Внутри класса опции доступны из свойства **aConfig:**

```
$this->aConfig['myoption'];
```

- Задайте шаблон для отображения формы и иконки драйвера, добавив новые сеты в файл **user_source_app.tpl.php**.

```
##--  { --##
<!--#set var="test_login_button" value="
     HTML, JavaScript код формы авторизации
    <input type=”hidden” name=”option” value=”##option##” />
"-->
##-- } --##

##--  { --##
<!--#set var="test_icon" value="<img src="/_img/test-icon.png"> "-->
##-- } --##
```

- Пропишите имя сета отображения формы драйвера в свойстве **loginButtonSet**, а имя сета отображения иконки драйвера в свойстве **iconSet**.

```
/**
 * Name of login button set
 *
 * @var string
 */
protected $loginButtonSet = 'test_login_button';

/**
 * Icon set.
 *
 * @var string
 */
protected $iconSet = 'test_icon';
```

- Реализуйте метод **getButton**, выводящий иконку драйвера

```
    public function getButton(){
      $option = $this->aConfig['myoption'];
      return parent::getButton(array( 'option' => $option ));
    }
```

- *Опционально: модифицируйте метод **dispatchVerify**, если ваш драйвер позволяет пользователю авторизоваться через него и если настройки авторизации по умолчанию вас чем-либо не устраивают.*

```
  /**
 * Dispatch Verify login action.
 *
 * @param array $aData  Array. Required keys: 'login'.
 * @return bool
 */
function dispatchVerify(array $aData = array()){

    // some custom logic

    return parent::dispatchVerify($aData);
}
```

Модификация для выполнения кастомных запросов

Существующая точка входа /ami_service.php позволяет выполнять любые нестандартные методы вашего драйвера. Для этого требуется сформировать POST или GET запрос, содержащий следующие параметры:

service = source_app - обязательный параметр
driver = ami_test - название драйвера
driver_action = verify – вызываемое действие

При этом, в теле класса необходимо объявить публичный метод с именем **dispatchДействие**, в нашем случае **dispatchVerify**.

http://127.0.0.1/ami_service.php?service=source_app&driver=ami_test&driver_action=verify&id=111&test_param=123

внутри своего метода вы можете получать параметры из запроса с использованием объекта **AMI_Request** (http://manual.amiro.ru/docs/api6/Environment/AMI_Request.html).

```
public function dispatchVerify()
{
	$oRequest = AMI::getSingleton('env/request');
	$testParam = $oRequest->get('test_param');

	// . . . . .

	return parent::dispatchVerify($aData);
}
```

Объединение систем авторизации с внешним приложением.

С версии 5.12.4 Amiro.CMS появилась возможность использовать схему единой авторизации CMS и различных внешних приложений. Данная схема подразумевает единую точку регистрации, единый набор пользователей и паролей, синхронизация данных пользователей при операции обновления.

В данном разделе будут продемонстрированы основные возможности единой авторизации на примере интеграции с форумом vBulletin 4.xx

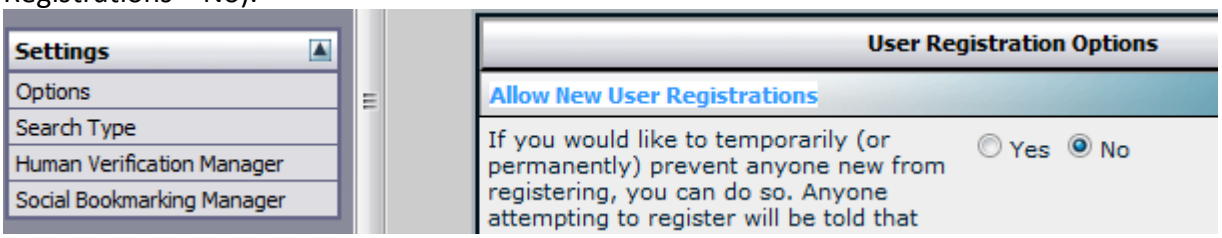
Объединение системы авторизации Amiro.CMS и сторонней системы на примере форума vBulletin 4.xx

Установка и настройка форума

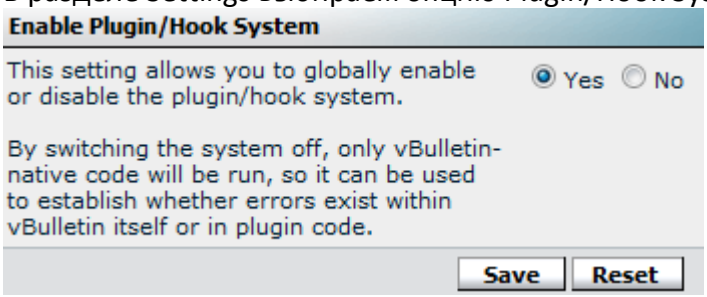
Предполагаем, что пользователь уже является владельцем дистрибутива и лицензионного ключа vBulletin, и в состоянии самостоятельно произвести инсталляцию форумного скрипта.

1) Создаем в корне сайта папку «forum» и устанавливаем в ней форум vBulletin 4.xx. (install/install.php)

2) В панели управления форума открываем раздел Settings, выбираем User Registration Options и запрещаем регистрацию пользователей на форуме (Allow New User Registrations = No).



3) В разделе Settings выбираем опцию Plugin/Hook System и включаем систему плагинов.



4) В разделе Settings выбираем опцию Server Settings and Optimization Options. Выставим значение опции Session IP Octet Length Check в значение 255.255.255.255. Это необходимо для корректной передачи сессии от драйвера клиенту.

На этом предварительная настройка форума завершена.

Подключение драйвера внешней авторизации

Для использования возможности объединения систем авторизации Amiro.CMS и форума vBulletin на стороне CMS необходимо подключить и инициализировать соответствующий драйвер внешней авторизации. Для этого добавим в файл _local/front_functions.php следующий код:

```
function EventInitBefore(&$vObject){
  AMI::getSingleton('extauth/drivers/vbulletin')->init(array(
    'forum_url'   => 'http://127.0.0.1/forum',
    'db_host'     => 'localhost',
    'db_username' => 'root',
    'db_password' => '',
    'db_database' => 'forum'
    'db_prefix'   => 'vb_',
  ));
  return true;
}
```

Здесь:
forum_url – URL по которому размещается форум
db_host, db_username, db_password, db_database – данные для доступа к БД форума (указывались при установке скрипта vBulletin).
db_prefix указывает на префикс таблиц vBulletin, если вы используете одну базу данных для форума и CMS. Если вы не указывали префикс таблиц при установке форума, эту опцию можно не передавать.

Дополнительно есть возможность указать параметры cookie_ttl – время жизни cookies форума в секундах (по умолчанию 3600) и cookie_path – путь размещения cookies (полный URL).

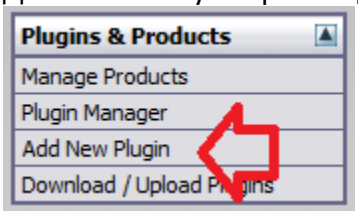
В результате, на события регистрации, изменения профиля, авторизации и выхода будут добавлены соответствующие обработчики, обращающиеся к скриптам vBulletin и выполняющие соответствующие действия на стороне форума.

Таким образом, на этом этапе, регистрация на стороне CMS добавит соответствующего пользователя на форуме, авторизация на стороне CMS одновременно авторизует пользователя на стороне форума, завершение сессии пользователя на стороне CMS завершит сессию на стороне vBulletin, и т.д.

Добавление хуков (hooks) форума

Для того чтобы настроить обратный процесс авторизации (авторизация на форуме влияет на авторизацию CMS) в панели управления форума необходимо добавить обработчики событий (хуки) для событий авторизации, выхода и обновления регистрационных данных.

Добавление хука производится через раздел Plugins & Products, опцию Add New Plugin.



Авторизация

Хук на событие login_process. Имя хука можно задать любое, но лучше использовать говорящие наименования, например, “Amiro.CMS Login”.

```
$username = $vbulletin->userinfo['username'];

AMI::saveGlobalScope('vB');
AMI::restoreGlobalScope('Amiro');

AMI_Registry::set('vb_acton', true);

$oSession = AMI::getSingleton('env/session');

$oSession->start();
$oSession->loginAsUser($username);
$oSession->store();

AMI::restoreGlobalScope('vB');
```

Здесь конструкция

```
AMI::saveGlobalScope('vB');
AMI::restoreGlobalScope('Amiro');
. . .
AMI::restoreGlobalScope('vB');
```

предназначена для изоляции глобальных переменных форума и CMS, для исключения возможного влияния их друг на друга. Глобальные переменные vBulletin сохраняются с именем vB, восстанавливаются глобальные переменные CMS, а по окончанию работы с API Amiro.CMS вновь восстанавливаются глобальные переменные форума.

```
AMI_Registry::set('vb_acton', true);
```

Это необходимо для предотвращения повторной отработки события на стороне CMS.

Прекращение авторизации

Хук на событие logout_process.

```
AMI::saveGlobalScope('vB');
AMI::restoreGlobalScope('Amiro');

$oSession = AMI::getSingleton('env/session');
$oSession->stop();

AMI::restoreGlobalScope('vB');
```

Обновление данных

Хук на событие profile_updatepassword_complete.

```
$username = $vbulletin->userinfo['username'];
$email = $vbulletin->GPC['email'];
$password = $vbulletin->GPC['newpassword'];

AMI::saveGlobalScope('vB');
AMI::restoreGlobalScope('Amiro');

AMI_Registry::set('vb_acton', true);

$oSession = AMI::getSingleton('env/session');

$oUsersModel = AMI::getResourceModel('users/table');
$oUsersItem = $oUsersModel->getItem();
$oUsersItem->loadByFields(array('login' => $username));

$oUsersItem->email = $email;
$oUsersItem->password = $password;
$oUsersItem->save();
$oUsersItem->savePassword(true);


AMI::restoreGlobalScope('vB');
```

Настройка для работы с форумом на поддомене

Для того чтобы CMS имела возможность использовать общую авторизацию с форумом, который расположен на поддомене, необходимо, чтобы скрипты поддомена имели доступ к файлам основного домена.

Для интеграции необходимо осуществить следующие настройки.

Настройка форума

По умолчанию, vBulletin выставляет cookie для текущего домена.

Необходимо в разделе Cookies and HTTP Options в опциях Cookie Domain – Suggested Settings выбрать имя основного домена.

Настройка CMS

В код инициализации драйвера в файле _local/front_functions.php необходимо добавить опцию cookie_path, значение которой соответствует полному URL основного сайта.

```
function EventInitBefore(&$vObject){
  AMI::getSingleton('extauth/drivers/vbulletin')->init(array(
     'forum_url'   => 'http://forum.mysite.ru/',
     'db_host'     => 'localhost',
     'db_username' => 'root',
     'db_password' => '',
     'db_database' => 'forum'
     'db_prefix'   => 'vb_',
     'cookie_path' => 'http://mysite.ru/',
  ));
  return true;
}
```

Создание собственного драйвера внешней авторизации

Драйвер авторизации есть наследник класса AMI_ExternalAuthDriver.

```
class My_ExternalAuthDriver extends AMI_ExternalAuthDriver {

}
```

Для корректной работы драйвера необходимо задать массив ожидаемых настроек:

```
private $aSettings = array(
   'my_option1' => 'default value 1',
   'my_option2' => 'default value 2',
   'my_option3' => 'default value 3'
);
```

И реализовать методы getInstance() и init():

```
public static function getInstance(){
      if(is_null(self::$oInstance)){
         self::$oInstance = new My_ExternalAuthDriver();
      }
      return self::$oInstance;
}

public function init(array $aSettings = array()){
      parent::init($aSettings);
      // Event handlers
      return true;
}
```

Далее создаются публичные методы класса отвечающие за обработку тех или иных событий на стороне CMS.

Например:

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

     // Your custom code

     return $aEvent;
}

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

     // Your custom code

     return $aEvent;
}
```

Эти обработчики подключаются в методе init():

```
AMI_Event::addHandler(
    'on_after_user_login',
    array($this, 'afterLogin'),
    AMI_Event::MOD_ANY);

AMI_Event::addHandler(
    'on_before_user_logout',
    array($this, 'beforeLogout'),
    AMI_Event::MOD_ANY);
```

Аналогичным образом создаются обработчики на события создания пользователя и обновления данных профиля.

Список доступных событий:

on_before_user_create
on_after_user_create
on_before_user_update
on_after_user_update
on_before_user_login
on_after_user_login
on_before_user_logout

Полученный драйвер желательно прописать в ресурсах как ‘extauth/driver/my’.