How it works...

Let’s start with the test case. Since we want to use a set of models, we are defining fixtures. A fixture set is put into the “database” each time the test method is executed.

We prepare unit tests for specifying how the behavior must work:

* First, we are testing a processing of a new model content. The behavior must convert the Markdown text from the source attribute to HTML and store the second one to the target attribute.
* Second, we are testing to update the content of the existing model. After changing the Markdown content and saving the model, we must get the updated HTML content.

Now let’s move to the interesting implementation details. In behavior, we can add our own methods, which will be mixed into the model that the behavior is attached to. Also, we can subscribe to the owner component events. We are using it to add an own listener:

public function events()

{

return [

ActiveRecord::EVENT\_BEFORE\_INSERT => 'onBeforeSave',

ActiveRecord::EVENT\_BEFORE\_UPDATE => 'onBeforeSave',

];

}

Now we can implement this listener:

public function onBeforeSave(Event $event)

{

if ($this->owner->isAttributeChanged($this->sourceAttribute))

{

$this->processContent();

}

}

In all the methods, we can use the owner property to get the object the behavior is attached to. In general, we can attach any behavior to our models, controllers, applications, and other components that extend the yii\base\Component class. Also, we can attach one behavior repeatedly to the model for processing different attributes:

class Post extends ActiveRecord {

public function behaviors() {

return [

[

|  |  |  |  |
| --- | --- | --- | --- |
|  | 'class' => MarkdownBehavior::className(), | | |
|  | ' sourceAttribute' | = > | ' description\_markdown |
| ], | 'targetAttribute' | = > | 'description\_html', |
| [ | 'class' => MarkdownBehavior::className(), | | |
|  | ' sourceAttribute' | = > | 'content\_markdown', |
| ], | 'targetAttribute' | = > | 'content\_html', |