Creating Events

Table of Contents

The Events system has been deprecated and replaced since Blesta version 4.3.0

The event system and its event handlers have been deprecated and replaced as of version 4.3.0. You should no longer use the deprecated event system from the 'Events' component in any of your custom code. The following describes the new event system.

Creating an Event

Creating an event simply sets a callback to be invoked if and only if during the execution of the current program such an event is triggered. This is useful if you require notification of certain actions throughout your application.

Registering an Event

Before registering an event you will need to setup an event listener:

Building an event listener

```
// Code that inherits from AppController or AppModel may retrieve an instance of the
\Blesta\Core\Util\Events\EventFactory from the container by calling $this->getFromContainer('util.events')
$eventFactory = $this->getFromContainer('util.events');
$eventListener = $eventFactory->listener();

// If you are working in a location that does not inherit from AppController or AppModel, you can load the
\Blesta\Core\Util\Events\EventFactory yourself
$eventFactory = new \Blesta\Core\Util\Events\EventFactory();
$eventListener = $eventFactory->listener();
```

Registering an event will allow your callback to be notified when such an event is triggered.

Your model or controller

```
$eventListener->register('EventName', [$this, 'callbackMethod']);
```

In the above example, when the "EventName" event is triggered the "callbackMethod" will be invoked on the object that registered the event. If you are registering one of the core events available on the Event Handlers page, the second argument to *register* (i.e. the callback) can be omitted and Blesta will use its predefined callback to perform all related event actions.

An event can be registered more than once

If you register the same event with the same callback more than once that callback will be executed multiple times each time the event is triggered.

Triggering an Event

Triggering an event invokes all callbacks that have been registered for that event. To trigger the "EventName" event invoke the Listener's *trigger* method and pass in an **\Blesta\Core\Util\Events\Common\EventInterface** that will be passed to each callback.

Somewhere in your code

```
$eventListener->trigger(
   $eventFactory->event('EventName', ['square', 'pink'])
);
```

Here's a complete example that will output "square pink".

/app/controllers/my_controller.php

```
<?php
class MyController extends AppController
{
    public function index()
    {
                // Register the 'EventName' and have it trigger our 'callbackMethod'
                $eventFactory = $this->getFromContainer('util.events');
        $eventListener = $eventFactory->listener();
        $eventListener->register('EventName', [$this, 'callbackMethod']);
        $eventListener->trigger($eventFactory->event('EventName', ['square', 'pink']));
        return false; // don't render a view
    }

    public function callbackMethod($event)
    {
        $params = $event->getParams();
        echo $params[0] . ' ' . $params[1]; // square pink
    }
}
?>
```

Returning Data From an Event

> **The same event is passed to all handlers**
>
> Keep in mind that setting a return value could override the return value set by a previous event handler listening to this event.

The **\Blesta\Core\Util\Events\Common\EventInterface** passed to each event callback contains setters and getters for both parameters (as seen above when triggering an event) and return values. Expanding on the example above, we can set our return value using the *\Blesta\Core\Util\Events\Common\EventInterface::setReturnValue()* method. This will output "square".

/app/controllers/my_controller.php

```
<?php
class MyController extends AppController {
    public function index()
    {
        $eventFactory = $this->getFromContainer('util.events');
        $eventListener = $eventFactory->listener();
        $eventListener->register('EventName', [$this, 'callbackMethod']);
        $eventListener->trigger($eventFactory->event('EventName', ['square', 'pink']));
        echo $event->getReturnValue(); // square
        return false; // don't render a view
    }

    public function callbackMethod($event)
    {
        $params = $event->getParams();
        $event->setReturnValue($params[0]);
    }
}
?>
```

Creating Events prior to Blesta version 4.3.0

> **The Events system has been deprecated and replaced since Blesta version 4.3.0**
>
> The event system described below, via the *Events* component, has been deprecated and replaced as of version 4.3.0. You should no longer use the deprecated event system in any of your custom code.

Creating an event simply sets a callback to be invoked if and only if during the execution of the current program such an event is triggered. This is useful if you require notification of certain actions throughout your application.

Registering an Event

Registering an event will allow your callback to be notified when such an event is triggered. Any controller or model that inherits from AppController or AppModel will already have access to the event handler and can immediately begin registering events.

Your model or controller

```
$this->Events->register("EventName", array($this, "callbackMethod"));
```

In the above example, when the "EventName" event is triggered the "callbackMethod" will be invoked on the object that registered the event. If you're working from a separate location that doesn't inherit from AppController or AppModel, you can load the Event handler manually.

Manually loading the Event handler

```
// Load the Event handler
Loader::load(COMPONENTDIR . "events" . DS . "events.php");
$this->Events = new Events();
```


An event can be registered more than once

If you register the same event with the same callback more than once that callback will be executed multiple times each time the event is triggered.

Triggering an Event

Triggering an event invokes all callbacks that have been registered for that event. To trigger the "EventName" event invoke the Events::trigger() method and pass in an instance of the **EventObject** that will be passed to each callback.

Somewhere in your code

```
$event = new EventObject("EventName", array("square", "pink"));
$this->Events->trigger($event);
```

Here's a complete example that will output "square pink".

/app/controllers/my_controller.php

```
<?php
class MyController extends AppController {

    public function index() {
        $this->Events->register("EventName", array($this, "callbackMethod"));
        $event = new EventObject("EventName", array("square", "pink"));
        $this->Events->trigger($event);
        return false; // don't render a view
    }

    public function callbackMethod(EventObject $event) {
        $params = $event->getParams();
        echo $params[0] . " " . $params[1];
    }
}
?>
```

Returning Data From an Event

The EventObject passed to each event callback contains setters and getters for both parameters (as seen above when Triggering an Event), and return values. Expanding on the example above, we can set our return value using the EventObject::setReturnVal() method. This will output "square".

/app/controllers/my_controller.php

```
<?php
class MyController extends AppController {
    public function index() {
        $this->Events->register("EventName", array($this, "callbackMethod"));
        $event = new EventObject("EventName", array("square", "pink"));
        $event = $this->Events->trigger($event);
        echo $event->getReturnVal();
        return false; // don't render a view
    }

    public function callbackMethod(EventObject $event) {
        $params = $event->getParams();
        $event->setReturnVal($params[0]);
    }
}
?>
```