# Events

## Overview

Events provide a simple observer implementation, allowing one to subscribe and listen for events in the application.

## The Event

When an event is triggered, it's identified by a unique name (e.g. system.onRoute), which any number of listeners might be listening to. An Event instance is also created and passed to all of the listeners.

### Naming Conventions

The unique event name can be any string, but optionally follows a few simple naming conventions:

- use only lowercase letters, numbers, dots (.) and underscores (_);
- prefix names with a namespace followed by a dot (e.g. kernel.);
- end names with a verb that indicates what action is being taken (e.g. request).

### Event Object

When the dispatcher notifies listeners, it passes an actual Event object to those listeners. The base Event class is very simple: it contains a method for stopping event propagation, methods for adding and removing arguments (i.e., data to be passed to the listeners), and a method for adding to the response.

Often times, data about a specific event needs to be passed along with the Event object so that the listeners have needed information.

```
// Creating an Event called "onSomething".
$event = new HubzeroEventsEvent('system.onDoSomething');

// Adding an argument named "foo" with value "bar".
$event->addArgument('foo', 'bar');
```

Arguments can be added (if not already existing), forcefully set, retrieved, or removed.

addArgument

Add an event argument, only if it is not existing.

setArgument

Set the value of an event argument. If the argument already exists, it will be overridden.

removeArgument

Remove an event argument.

getArgument

Get an event argument value.

hasArgument

Tell if the given event argument exists.

```
class EventListener
{
    public function onDoSomething($event)
    {
        // Check that the event has the necessary 'foo' argument
        if ( ! $event->hasArgument('foo'))
        {
            return;
        }

        // Get the 'foo' argument
        $foo = $event->getArgument('foo');
    }
}
```

## The Dispatcher

The dispatcher is the central object of the event dispatcher system. In general, a single dispatcher is created, which maintains a registry of listeners. When an event is triggered via the dispatcher, it notifies all listeners registered with that event:

```
$dispatcher = new HubzeroEventsDispatcher();
```

### Registering Listeners

Registering an event listener can be done simply by passing the listener object to the addListener method.

```
$dispatcher->addListener(new SystemListener);
```

By default, the listener will be registered to all events matching its method names. If the listener contains methods that should not be registered, a defined list of events may be passed.

```
$dispatcher->addListener(
    new SystemListener,
    array(
        'onBeforeRoute' => HubzeroEventsPriority::NORMAL,
        'onAfterRoute' => HubzeroEventsPriority::NORMAL,
    )
);
```

In the above example, the SystemListener object is registered as an even listener along with a specified list of events that it listens to. When a defined list of events is passed, a listener's priority for a given Event is also specified.

When a Listener is added without specifying the event names, it is registered with a NORMAL priority to all events. If some listeners have the same priority for a given event, they will be called in the order they were added to the Dispatcher.

It is also possible to register a closure or anonymous function as an event listener:

```
$dispatcher->addListener(
    function($event) {
        $foo = $event->getArgument('foo');
        // ... do cool things here ...
    },
    array(
        'onBeforeRoute' => HubzeroEventsPriority::NORMAL
    )
);
```

## Triggering Events

Once listeners and events have been registered, the events can be triggered. The listeners will be called in a queue according to their priority for that Event.

```
// Triggering the onBeforeRoute Event.
$dispatcher->trigger('onBeforeRoute');
```

The trigger method not only accepts an event name but can also accept a custom Event object.

```
// Creating an event called "onBeforeRoute" with a "foo" argument.
$event = new HubzeroEventsEvent('onAfterSomething');
$event->setArgument('foo', 'bar');

$dispatcher->trigger($event);
```

Arguments or data to be passed to the listener can be specified in an array as a second parameter to the trigger.

```
// Triggering the onBeforeRoute Event.
$dispatcher->trigger('onBeforeRoute', array($foo, $bar));
```

In the example above, the two variables $foo and $bar are added as arguments to the event object and passed to the listener. This is functionally equivalent to the following:

```
// Creating an event called "onBeforeRoute" with a "foo" argument.
$event = new HubzeroEventsEvent('onAfterSomething');
$event->setArgument('foo', $foo);
$event->setArgument('bar', $bar);

$dispatcher->trigger($event);
```

## Stopping Events

In some cases, it may make sense for a listener to prevent any other listeners from being called. That is, the listener needs to be able to tell the dispatcher to stop all propagation of the event to future listeners. This can be accomplished from inside a listener by calling the stop() method on the event:

```
class SystemListener
{
    public function onBeforeRoute(Event $event)
    {
        // Stopping the Event propagation.
        $event->stop();
    }
}
```

When stopping the Event propagation, the next listeners in the queue won't be called.

It is possible to detect if an event was stopped by using the isStopped() method which returns a boolean value:

```
class SystemListener
{
    public function onBeforeRoute(Event $event)
    {
        // Stopping the Event propagation.
        $event->stop();

        if ($event->isStopped())
        {
            //...
        }
    }
}
```