# Loading

## Triggering Events

Plugins are lazy-loaded by default, which means they must be imported and registered with the event dispatcher on a "as-needed" basis. This can be accomplished by using dot-notation when triggering the event (more on that later) or by manually importing the necessary plugin group:

```
Plugin::import('groups');
```

The above line will import all published plugins of the type "groups"—that is, all plugins in /plugins/groups—and register them as event listeners with the dispatcher.

To fire an event, one may use the Event facade, passing an instance of the event to the trigger method. The trigger method will dispatch the event to all of its registered listeners:

```
// Import the "media" plugins
Plugin::import('media');

// Trigger the event
$results = Event::trigger('onAlbumAddedToLibrary', array($artist, $tit
le));
```

Here we have triggered the event 'onAlbumAddedToLibrary' and passed in the artist name and title of the album. All plug-ins will receive these parameters, process them and optionally pass back information. The trigger method will always return an array.

Although relatively short, the above code example can be simplified even further by using dot-notation to combine the plugin group and event name into one:

```
// Load the plugin group "media" and trigger the event
$results = Event::trigger('media.onAlbumAddedToLibrary', array($artist
, $title));
```

Here, the trigger method recognizes dot-notation being used and extracts the plugin group from the string, imports said plugin group, and registers them with the event dispatcher before

triggering the event. For those concerned about performance, it should be noted the importing of plugins *will only happen once*.

**Note:** One thing to notice about the trigger method is that there is nothing defining which group of plug-ins should be *notified*. In actuality, all plug-ins that have been loaded are notified regardless of the group they are in. So, it's important to be sure that event names do not conflict with any other plugin group's event name.

## Stopping an Event

Sometimes, a plugin may need to prevent any further plugins from responding to an event. In such cases, the event loop can be halted.

When an event is triggered, an event object is created to track responders, pass data, and collect responses from listeners. For anonymous functions, this event object is passed as the only argument. For legacy plugins, the object is attached as a public property to the plugin and can be accessed by calling $this->event.

So, stopping an event is done by calling stop on the event object.

```
<?php
class plgSystemExample extends Plugin
{
    public function onAfterRoute()
    {
        // ... some logic here ...

        $this->event->stop();
    }
}
```