ModuleFields

The ModuleFields class is automatically made available to any module. To begin using it simply initialize the object within your module method.

Module methods that must return a ModuleFields object include:

- getPackageFields
- getAdminAddFields
- getClientAddFields
- getAdminEditFields

Creating ModuleFields

Creating fields using the ModuleFields class allows you to define a set of input fields that may be displayed in a variety of contexts, without worrying about how those fields are contained. There are three aspects of creating a field, described below, that include: creating the field label, creating the field, and attaching the field to the label.

Creating Labels

Creating a label will return a ModuleField object.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getPackageFields($vars=null) {
        // Create the label
        $fields = new ModuleFields();
        $my_label = $fields->label("My Field Label", "my_field_label_id");

        $fields->setField($my_label);
        return $fields;
    }
}
?>
```

Creating Fields

As with creating a label, creating a field will return a ModuleField object. The example below uses the **ModuleFields::fieldText()** method, but others exist for creating radio buttons, textarea fields, and everything else you'll need.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getPackageFields($vars=null) {
        // Create the label
        $fields = new ModuleFields();
        $my_label = $fields->label("My Field Label", "my_field_id");

        // Create the field
        $my_field = $fields->fieldText("my_field_name", "default value", array('id' => "my_field_id"));

        $fields->setField($my_label);
        return $fields;
    }
}
?>
```

Attaching a Fields to a Label

A field may be attached to a label using the **ModuleField::attach()** method, or, instead, attach a label to a field using **ModuleField::setLabel()**.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getPackageFields($vars=null) {
        // Create the label
        $fields = new ModuleFields();
        $my_label = $fields->label("My Field Label", "my_field_id");

        // Create the field
        $my_field = $fields->fieldText("my_field_name", "default value", array('id' => "my_field_id"));

        // Attach the field to the label
        $my_label->attach($my_field);

        // OR, attach the label to the field, instead
        // $my_field->setLabel($my_label);

        $fields->setField($my_label);
        return $fields;
    }
}
?>
```

Adding tooltips to a Label

Tooltips can be created using the **ModuleFields::tooltip()** method, then attached using **ModuleField::attach()**. In the example below we've eliminated unnecessary variable declarations to give a more concise example.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getPackageFields($vars=null) {
        // Create the label
        $fields = new ModuleFields();
        $my_label = $fields->label("My Field Label", "my_field_id");

        // Create and attach the field to the label
        $my_label->attach($fields->fieldText("my_field_name", "default value", array('id' => "my_field_id")));

        // Attach a tooltip to the label
        $my_label->attach($fields->tooltip("This is my tooltip"));

        $fields->setField($my_label);
        return $fields;
    }
}
?>
```

⚠ **Tooltips may only be set on top level labels**

A tooltip may not be attached to a label that is, itself, attached to a field.

Multiple tooltips may be created per label

... but that would be silly.

Complex Fields

Blesta supports ModuleFields up to two levels deep. This allows you to easily create a set of checkboxes or radio buttons associated with a single field.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getPackageFields($vars=null) {
        // Create the label
        $fields = new ModuleFields();
        $my_label = $fields->label("My Field Label", "my_field_id");

                // Create a field label displayed next to the checkbox
                $field_label = $fields->label("My Value", "my_field_my_value_id");

        // Create and attach the field to the label, set as checked (3rd param) if necessary
        $my_label->attach($fields->fieldCheckbox("my_field_name", "my_value", (isset($vars->my_field_name) &&
$vars->my_field_name == "my_value"), array('id' => "my_field_my_value_id"), $field_label));

        // Attach a tooltip to the label
        $my_label->attach($fields->tooltip("This is my tooltip"));

        $fields->setField($my_label);
        return $fields;
    }
}
?>
```

Creating HTML content

The ModuleFields class supports HTML content as well. You may choose to set nothing but HTML content, but the best practice is to set fields using the method mentioned in the section above and only set javascript and that sort of thing as HTML content.

To set HTML content use the ModuleFields::setHtml() method.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getPackageFields($vars=null) {
        $fields = new ModuleFields();
        $fields->setHtml("
            <script type=\"text/javascript\">alert('ok!');</script>
        ");

    }
}
?>
```

Overriding Package/Service Fields

Generally it is a good idea to name your fields such that they work within the scope of your own module. For example, if your module requires a field named "term", naming it instead "my_module_term" will prevent conflicts with fields used by the system. However, in some cases you may want to allow your module to set a service field directly. One such example is the quantity field (qty). Here's how you can set a quantity field when creating a service.

/components/modules/my_module/my_module.php

```
<?php
class MyModule extends Module {

    ...

    public function getAdminAddFields($package, $vars=null) {
        Loader::loadHelpers($this, array("Html"));

        $fields = new ModuleFields();

                // Create qty label
        $qty = $fields->label("Quantity", "my_qty");
        // Create qty field and attach to qty label
        $qty->attach($fields->fieldText("qty", $this->Html->ifSet($vars->qty), array('id' => "my_qty")));
        // Set the label as a field
        $fields->setField($qty);

    }
}
?>
```