Widgets

A widget is an interface element that can contain just about anything. To help create widgets we've designed the **Widget**, or **WidgetClient** helper, which is automatically loaded for all views created in a controller that inherits (either directly or indirectly) from AppController. The Widget helper is loaded for the staff interface, and the WidgetClient helper is loaded for the client interface. The two are almost identical, except that the WidgetClient helper doesn't provide some additional options. In all of the examples below simply replace Widget with WidgetClient if working in the client interface.

The Structure

Every widget needs at least two things: a beginning, and an end.

/plugins/my_plugin/views/default/client_main.pdt

```
<?php
$this->Widget->clear();
$this->Widget->create();

#
# TODO: Place widget contents here
#

$this->Widget->end();
?>
```

In the example above we do three things:

1. On line 2 we clear the widget using the **clear()** method. This allows us to ensure that our widget is in its default state.
2. On line 3 we begin the widget with **create()**.
3. On line 9 we end the widget with **end()**.

The **create()** method accepts a few parameters, described below:

- **$title** - The title to display for the widget window
- **$attributes** - An array of HTML attributes to set for the widget's primary container.
- **$render** - The are of the widget to render. This is necessary when renderings widgets via AJAX, so you can control the content that's replaced. Acceptable values include:
    - 'full' - The default value, will render the entire widget.
    - 'content_section' - The full content including navigation (everything excluding box frame and title section). Use 'inner_content' if using WidgetClient.
    - 'common_box_content' - The content only ('content_section' excluding the navigation). Use 'inner' if using WidgetClient.

/plugins/my_plugin/views/default/client_main.pdt

```
<?php
$this->Widget->clear();
$this->Widget->create("My Widget Title", array('id' => "my_widget_id", 'class' => "my_widget_class"), "full");

#
# TODO: Place widget contents here
#

$this->Widget->end();
?>
```

Window Decorations

Window decorations are set on the title bar of the widget. There are currently three types of window decorations: **arrow**, **setting**, and **full_screen**. The arrow decorations allows the widget to be minimized and restored, while the setting decoration may be linked another page or URI loaded via AJAX. The full_screen decoration adds a button which allows this widget to expand to the full width of the browser window.

/plugins/my_plugin/views/default/client_main.pdt

```
<?php
$this->Widget->clear();
// Allow the widget to be minimized
$this->Widget->setWidgetButton("arrow");
// Allow this widget's setting to be updated using ClientMain::settings()
$this->Widget->setWidgetButton(array('class' => "setting ajax", 'href' => $this->base_uri . "widget/my_plugin
/client_main/settings/"));
$this->Widget->create("My Widget Title", array('id' => "my_widget_id", 'class' => "my_widget_class"), "full");

#
# TODO: Place widget contents here
#

$this->Widget->end();
?>
```

You'll notice on line 6, above, we've set both a "setting" and "ajax" class for our window decoration. The "setting" class works just line the "arrow" parameter on line 4, except it renders a different icon. The real difference here is the addition of the "ajax" class. What this does is tell Blesta to fetch the URI specified in the 'href' attribute via AJAX. Blesta will automatically request the data and perform the replacement of the data in the DOM. Therefore, the response of the request must be properly formatted. To do that, simply invoke the **AppController::renderAjaxWidgetIfAsync()** method.

/plugins/my_plugin/views/default/client_main.pdt

```
<?php
class ClientMain extends MyPluginController {
        ...
        public function settings() {

                $this->set("var1", "hello");
                $this->set("var2", "world!");

                return $this->renderAjaxWidgetIfAsync(false);
        }
}
?>
```

The renderAjaxWidgetIfAsync() method will automatically determine the format of the response and act accordingly.

Tabs and Links

Widgets may set tabs or links to appear at the top of the widget to swap content.

When tabs are set, each must include three attributes, namely: **name**, **current**, and **attributes**. The name is the display name of the tab. Current is whether or not the tab is currently selected, and attributes are additional attributes to include on the anchor tag of the tab, such as the href.

```
<?php
        $tabs = array(
                array('name'=>"Settings", 'attributes'=>array('href'=>$this->base_uri . "widget/my_plugin
/client_main/settings/")),
                array('name'=>"Manage", 'current'=>true, 'attributes'=>array('href'=>$this->base_uri . "widget
/my_plugin/client_main/manage/"))
        );

        $this->Widget->clear();
        $this->Widget->setTabs($tabs);
        $this->Widget->create("My Widget Title", true));

        #
        # TODO: Place widget contents here
        #

        $this->Widget->end();
?>
```

Links may be used in place of tabs, and may accompany link buttons. Links are displayed at the top left of a widget while link buttons are displayed at the top right. Links may also specify an "ajax" class to allow the widget content to be replaced via AJAX content from the specified href.

```
<?php
        $links = array(
                array('name'=>"Settings", 'current'=>true, 'attributes'=>array('href'=>$this->base_uri . "widget
/my_plugin/client_main/settings/", 'class'=>"ajax")),
                array('name'=>"Manage", 'attributes'=>array('href'=>$this->base_uri . "widget/my_plugin
/client_main/manage/", 'class'=>"ajax"))
        );
        $link_buttons = array(
                array('name'=>"Add Setting", 'attributes'=>array("href"=>$this->Html->safe($this->base_uri .
"widget/my_plugin/client_main/settings/add/")))
        );

        $this->Widget->clear();
        $this->Widget->setLinks($links);
        $this->Widget->setLinkButtons($link_buttons);
        $this->Widget->create("My Widget Title", true));

        #
        # TODO: Place widget contents here
        #

        $this->Widget->end();
?>
```

Filters

As of v4.10.0 it is possible to add filters to widgets, these can be used to filter the data on a table or any information presented within the widget. The selected filters will be posted and their values will be available from the **$this->post** variable in the controller.

The **setFilters()** method accepts a few parameters, described below:

- **$filters** - An instance of InputFields, containing the fields display in the widget form.
- **$uri** - The URI where the form fields will be submitted.
- **$show_filters** - Wether to show the filtering form on page load, false by default.

The list of input fields can be defined within the controller using a private method, These are defined in the same way as the input fields on Modules.

```php
<?php
use Blesta\Core\Util\Input\Fields\InputFields;

class MyController extends MyPluginController {
        ...

        public function myWidget()
        {
                // Set filters from post input
        $post_filters = [];
        if (isset($this->post['filters'])) {
            $post_filters = $this->post['filters'];
            unset($this->post['filters']);
        }

                ...

                // Set the input field filters for the widget
        $filters = $this->getFilters($post_filters);
        $this->set('filters', $filters);
        $this->set('filter_vars', $post_filters);
        }

        private function getFilters(array $vars)
        {
                $filters = new InputFields();

                // Create "Service Name" label
        $service_name = $filters->label('Service Name', 'service_name');
                // Create "filters[service_name]" input field
        $service_name->attach(
            $filters->fieldText(
                'filters[service_name]',
                isset($vars['service_name']) ? $vars['service_name'] : null, // Pre-populate the field if it is
defined in $vars
                [
                    'id' => 'text_input',
                    'class' => 'form-control',
                    'placeholder' => 'Service Name'
                ]
            )
        );
        $filters->setField($service_name);

                // Set control panels list
                $control_panels = [
                        'cpanel' => 'cPanel',
                        'plesk' => 'Plesk'
                ];
                // Create "Control Panel" label
        $control_panel = $filters->label('Control Panel', 'control_panel');
                // Create "filters[control_panel]" select field
        $control_panel->attach(
            $filters->fieldSelect(
                'filters[control_panel]',
                $control_panels,
                isset($vars['control_panel']) ? $vars['control_panel'] : null,
                ['id' => 'control_panel', 'class' => 'form-control']
            )
        );
        $filters->setField($control_panel);

                // You can set HTML code as well!
        $fields->setHtml('
            <script type="text/javascript">alert("It Works!");</script>
        ');

                return $filters;
        }
}
```

After defining the filters in the controller, they can be added to the Widget from the view.

/plugins/my_plugin/views/default/client_mywidget.pdt

```
<?php
$this->Widget->clear();
$this->Widget->create();
$this->Widget->setFilters($filters, $this->base_uri . 'plugin/my_plugin/mywidget/', !empty($filter_vars));

#
# TODO: Place widget contents here
#

$this->Widget->end();
?>
```

Widgets and Plugins

Widgets can appear anywhere within your plugin, but Blesta reserves a few Plugin Actions especially for rendering widgets. These actions will automatically load your widgets to display in various locations (if the Staff member has enabled the widget to appear). This allows plugins to display custom data/forms on pages that plugins can't otherwise access. For more information on registering your widgets with these actions see the Plugin Actions section of this manual.

Custom Styles

Style Sheets

You can load custom style sheets in your widget simply by using the **setStyleSheet()** method. You may call it as many times as necessary.

/plugins/my_plugin/views/default/client_main.pdt

```
<?php
$this->Widget->clear();
$this->Widget->setStyleSheet($this->view_dir . "css/styles.css");
$this->Widget->create("My Widget Title", array('id' => "my_widget_id", 'class' => "my_widget_class"), "full");

#
# TODO: Place widget contents here
#

$this->Widget->end();
?>
```

Theming

You can enable links in your widgets to be themed by Blesta by adding the "override" class to each link.

Allow the theme to override the color of your links

```
...
<a class="override" href="/">Click me!</a>
...
```