How do I add a custom price to a product with a module?

In this mini tutorial we will add some checkboxes to the product page that when checked add an additional charge to the product. The additional charge will be shown dynamically on the product page as it changes or the product's price changes (due to different variants being selected).

This mini tutorial assumes that you have a module setup or you’re happy working from the app service provider (using the boot method). You can see how to set up a module here (link).

Adding the Prices Data

To get started we will add a `$prices` array to our module service provider that will be the source of truth for our extra prices. We are using a simple array instead of database data due to this being a mini tutorial. The `$prices` array will have id, name, and price (this will also have inc and ex values and be in pence) keys.

```
<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
   protected $prices = [
       [
           'id' => 1,
           'name' => 'Pay £1 more',
           'price' => [
               'inc' => 100,
               'ex' => 83.33,
           ],
       ],
```

```
        [
            'id' => 2,
            'name' => 'Pay £5 more',
            'price' => [
                'inc' => 500,
                'ex' => 416.6,
            ],
        ],
        [
            'id' => 3,
            'name' => 'Pay £10 more',
            'price' => [
                'inc' => 1000,
                'ex' => 833.33,
            ],
        ],
    ];

    public function setup()
    {
        //
    }
}
```

Creating a Javascript View and Adding the Prices Data and Javascript to the Product Page

Now we're going to create a Twig file called javascript where we will put the frontend javascript that will be used on the product page. This view is registered using the `$this->loadViewsFrom` method in the module service providers `setup` method.

Next we're going to use the `Aero\Store\Http\Responses\ProductPage` response builder to extend the product page. We will inject the $prices data using the setData method and then we'll use the `Aero\Store\Pipelines\ContentForBody` pipeline to add our javascript view to the product page.

```php
<?php

namespace Acme\MyModule;

use Aero\Common\Providers\ModuleServiceProvider;
use Aero\Store\Http\Responses\ProductPage;
use Aero\Store\Pipelines\ContentForBody;

class ServiceProvider extends ModuleServiceProvider
{
   protected $prices = [
       [
           'id' => 1,
           'name' => 'Pay £1 more',
           'price' => [
               'inc' => 100,
               'ex' => 83.33,
           ],
       ],
       [
           'id' => 2,
           'name' => 'Pay £5 more',
           'price' => [
               'inc' => 500,
               'ex' => 416.6,
           ],
       ],
       [
           'id' => 3,
           'name' => 'Pay £10 more',
           'price' => [
               'inc' => 1000,
               'ex' => 833.33,
           ],
       ],
   ];

   public function setup()
   {
```

```
        $this->loadViewsFrom(__DIR__.'/../resources/views', 'my-module');

        ProductPage::extend(function (ProductPage $page) {
            $page->setData('extra_prices', $this->prices);

            ContentForBody::extend(function (&$content) {
                $content .= view('my-module::javascript');
            });
        });
    }
}
```

Adding the Extra Prices to the Product Page

In the `product.twig` file we're going to add some Vue code to show the total price and the total extra price. Then we'll add some Twig code to loop over the `extra_prices` variable that we injected into the product page earlier. This loop will display the name as a label and then add a checkbox with a value of the extra price id and some data attributes for the price inc and ex values.

```
Total Price
<p v-text="total_price.inc"></p>

Total Extra Price
<p v-text="additional_prices['extra-prices'].inc" v-if="additional_prices['extra-
prices']"></p>

{% for extra in extra_prices %}
    <div>
        <input type="checkbox" id="extraPrice{{ extra.id }}" value="{{ extra.id }}" data-extra-
price-inc="{{ extra.price.inc }}" data-extra-price-ex="{{ extra.price.ex }}">
        <label for="extraPrice{{ extra.id }}"> {{ extra.name }}</label>
    </div>
{% endfor %}
```

Coding the Javascript for the Product Page

Now we're going to add javascript code to the javascript twig view we previously created. This javascript code will be responsible for setting the additional price when the checkboxes are checked and sending any checked extra prices to the server when the product is added to the cart.

To update the additional price when the checkboxes are checked we'll set some code to be executed on the `product.loaded` Aero Event. This code will loop over all input checkbox elements that have the `data-extra-price-inc` attribute and add an input event listener. We'll add a `updateAndSetAdditionalPrices` function that will be executed when the input changes and also initially once the product has loaded (in case you have some extra prices that are checked by default). This function loops over all input checked checkboxes that have the `data-extra-price-inc` attribute and adds up their inc and ex prices. Then it runs the product.set-additional-price Aero Event to tell Aero the new total additional price.

To send the checked extra prices to the server we'll set some code to be executed on the `product.add-to-cart` Aero Event. This code loops over all input checked checkboxes that have the `data-extra-price-inc` attribute and adds their values (the extra price id) to an array. This array is then added to the payload that will be sent to the server.

```
<script>
   window.AeroEvents.on('product.loaded', function () {
       var extraPriceElements = document.querySelectorAll('input[type="checkbox"][data-extra-price-inc]');

       for (var i = 0; i < extraPriceElements.length; i++) {
           extraPriceElements[i].addEventListener('input', updateAndSetAdditionalPrices);
       }

       function updateAndSetAdditionalPrices() {
           var price = { key: 'extra-prices', inc: 0, ex: 0 };
           var extraPriceElements = document.querySelectorAll('input[type=checkbox]:checked[data-extra-price-inc]');

           for (var i = 0; i < extraPriceElements.length; i++) {
               price.inc += parseInt(extraPriceElements[i].dataset.extraPriceInc);
               price.ex += parseInt(extraPriceElements[i].dataset.extraPriceEx);
           }
```

```
            window.Aero.runEvent('product.set-additional-price', price);
        }

        updateAndSetAdditionalPrices();
    });

    window.AeroEvents.on('product.add-to-cart', function (data) {
        var extras = [];
        var extraPriceElements = document.querySelectorAll('input[type=checkbox]:checked[data-extra-price-inc]');

        for (var i = 0; i < extraPriceElements.length; i++) {
            extras.push(extraPriceElements[i].value);
        }

        data.extras = extras;

        return data;
    });
</script>
```

Adding any Selected Extra Prices to the Cart Item

We'll add some code in the module service provider below the current code that extends the `Aero\Store\Pipelines\CartItemBuilder` and gets the extra price ids from the request, maps them to the relevant extra price, filters out any that are null (because they were not valid extra price ids), and then adds a `Aero\Cart\CartItemOption` to the `Aero\Cart\CartItem` with the extra price details.

```
<?php

namespace Acme\MyModule;

use Aero\Cart\CartItem;
use Aero\Cart\CartItemOption;
use Aero\Common\Providers\ModuleServiceProvider;
use Aero\Store\Http\Responses\ProductPage;
```

```
use Aero\Store\Pipelines\CartItemBuilder;
use Aero\Store\Pipelines\ContentForBody;

class ServiceProvider extends ModuleServiceProvider
{
   protected $prices = [
       [
           'id' => 1,
           'name' => 'Pay £1 more',
           'price' => [
               'inc' => 100,
               'ex' => 83.33,
           ],
       ],
       [
           'id' => 2,
           'name' => 'Pay £5 more',
           'price' => [
               'inc' => 500,
               'ex' => 416.6,
           ],
       ],
       [
           'id' => 3,
           'name' => 'Pay £10 more',
           'price' => [
               'inc' => 1000,
               'ex' => 833.33,
           ],
       ],
   ];

   public function setup()
   {
       $this->loadViewsFrom(__DIR__.'/../resources/views', 'my-module');

       ProductPage::extend(function (ProductPage $page) {
           $page->setData('extra_prices', $this->prices);

           ContentForBody::extend(function (&$content) {
               $content .= view('my-module::javascript');
```

```
            });
        });

        CartItemBuilder::extend(function (CartItem $item) {
            collect(request()->input('extras', []))->map(function ($extra) {
                return collect($this->prices)->firstWhere('id', $extra);
            })->filter()->each(function ($extra) use ($item) {
                $option = CartItemOption::create($extra['name']);
                $option->setPriceInc($extra['price']['inc']);

                $item->addOption($option);
            });
        });
    }
}
```

Revision #1
Created 2026-09-02 13:27:58 UTC by Michael Price
Updated 2026-09-02 13:27:58 UTC by Michael Price