

Order Management

- [Orders](#)

- [How do I block orders?](#)
- [How do I unblock orders?](#)
- [How do I customize the order reference?](#)
- [What is the order item builder?](#)
- [How do I add to the order activity timeline?](#)
- [How do I remove the manufacturers name from the order items name?](#)

- [Shipping](#)

- [How do I set a custom order validator for shipping methods?](#)
- [How do I set a custom order validator for shipping rates?](#)
- [How do I set a custom order validator for shipping rates \(for subscriptions\)?](#)
- [How do I set a custom price totalizer for shipping rates?](#)
- [How do I set a custom applicable items filter for shipping rates?](#)
- [How do I set a custom item weight totalizer for shipping rates?](#)
- [How do I set a custom item volume totalizer for shipping rates?](#)
- [How do I set a custom item quantity totalizer for shipping rates?](#)
- [How do I set a custom item price totalizer for shipping rates?](#)

- [Subscriptions](#)

- [What is the subscription order builder?](#)

Orders

How do I block orders?

Orders can be blocked by adding a blocker when some condition is met.

For example, you can prevent potentially fraudulent orders from proceeding to payment by checking the order's email domain and IP address against configured blocked values.

First, define some settings:

```
\Aero\Common\Facades\Settings::group('checkout', static function
(\Aero\Common\Settings\SettingGroup $group) {
    $group->array('blocked_email_domains')->section('Fraud');
    $group->array('blocked_ip_addresses')->section('Fraud');
});
```

An `is_fraudulent` getter can then be macro'ed onto the order model:

```
\Aero\Cart\Models\Order::macro('getIsFraudulentAttribute', function () {
    $orderEmailDomain = Str::after($this->email, '@');

    return in_array($orderEmailDomain, setting('checkout.blocked_email_domains'))
        || in_array($this->ip, setting('checkout.blocked_ip_addresses'));
});
```

Middleware can then be added to the checkout payment page to block the order if fraud has been detected:

```
\Aero\Checkout\Http\Responses\CheckoutPaymentPage::middleware(function
(\Illuminate\Http\Request $request, \Closure $next) {
    $order = \Aero\Cart\Facades\Cart::order();

    if ($order->is_fraudulent) {
        $order->setBlocker('fraud', [
            'reason' => 'We suspect that this order is fraudulent. Please contact our support
team if you have any questions.',
        ]);
    }
});
```

```
        return $next($request);
    });
```

The checkout payment page can then be extended to set a custom view to display the blocker reason:

```
\Aero\Checkout\Http\Responses\CheckoutPaymentPage::extend(function
(\Aero\Checkout\Http\Responses\CheckoutPaymentPage $page, \Closure $next) {
    $blocker = $page->cart->order()->getBlocker('fraud');

    if ($blocker) {
        $page->setData('blocker', $blocker);
        $page->getData('sections')->put('payment_method', 'acme::error');
    }

    return $next($page);
});
```

The `acme::error` view:

```
<div class="payment-method checkout__section">
    <div class="checkout__error">{{ blocker.reason }}</div>
</div>
```

Parameters

Parameter	Type	Description
<code>\$key</code>	<code>string</code>	The key of the blocker to set.
<code>\$value</code>	<code>mixed</code>	The value of the blocker to set.

How do I unblock orders?

You can remove an existing blocker from an order using the `unsetBlocker` method.

For example, you can make it so some customer groups are exempt from being blocked

```
\Aero\Common\Facades\Settings::group('checkout', static function
(\Aero\Common\Settings\SettingGroup $group) {
    $group->eloquent('exempt_customer_groups', \Aero\Account\Models\CustomerGroup::class)-
    >multiple()->dontLoadModels()->section('Fraud');
});
```

Middleware can then be added to the checkout payment page to unblock the order for customers of that group:

```
\Aero\Checkout\Http\Responses\CheckoutPaymentPage::middleware(function
(\Illuminate\Http\Request $request, \Closure $next) {
    $order = \Aero\Cart\Facades\Cart::order();

    $customerGroupId = optional($order->customer)->customer_group_id ?? null;

    if (in_array($customerGroupId, setting('checkout.exempt_customer_groups'))) {
        collect($order->blockers)->keys()
            ->each(function (string $key) use ($order) {
                $order->unsetBlocker($key);
            });
    }

    return $next($request);
});
```

Parameters

Parameter	Type	Description
<code>\$key</code>	<code>string</code>	The key of the blocker to unset.

How do I customize the order reference?

The order reference generator can be customized to control the format of order references that are generated when an order is created.

For example, to use the order reference prefix followed by the order ID:

```
\Aero\Cart\Models\Order::setReferenceGenerator(function (\Aero\Cart\Models\Order $order) {  
    $prefix = setting('order_reference_prefix');  
  
    return "{$prefix}{$order->getKey()}";  
});
```

The generator receives the order and should return a string to be used as the order reference.

Parameters

Parameter	Type	Description
\$order	\Aero\Cart\Models\Order	The order to generate a reference for.

What is the order item builder?

The order item builder is run whenever a cart item is transformed into an order item. It builds the order item data from the cart item.

For example, you can add a custom attribute to the order item:

```
\Aero\Cart\OrderItemBuilder::extend(function (\Aero\Cart\OrderItemBuilder $builder, \Closure $next) {
    $cartItem = $builder->item();

    $builder->setAttributes([
        'acme' => $cartItem->meta->get('acme'),
    ]);

    return $next($builder);
});
```

Parameters

Parameter	Type	Description
<code>\$builder</code>	<code>\Aero\Cart\OrderItemBuilder</code>	The order item builder.
<code>\$next</code>	<code>\Closure</code>	The next step in the order item builder.

Methods

Method	Description
<code>item()</code>	Returns the cart item being transformed.

<code>order()</code>	Returns the order that the order item will be added to once transformed.
<code>setAttribute(string \$key, \$value)</code>	Sets an attribute to be added to the order item.
<code>setAttributes(array \$attributes)</code>	Sets attributes to be added to the order item (merges them with existing attributes).
<code>getAttribute(string \$key)</code>	Returns the attribute corresponding to <code>\$key</code> or <code>null</code> if it doesn't exist
<code>getAttributes()</code>	Returns the array of attributes to be set on the order item

How do I add to the order activity timeline?

To add something to the order activity timeline you must create a class that extends

`\Aero\Admin\Activity`:

```
class OrderFlagsEditActivity extends \Aero\Admin\Activity
{
    protected $view = 'admin::activity.edit-flags';

    protected $key = 'order-flags-activity';

    protected $name = 'Order Flag Edits';

    protected $flags;

    public static function create(\Carbon\Carbon $time, ?\Aero\Admin\Models\Admin $admin,
\Illuminate\Support\Collection $flags): self
    {
        return (new self($time))->admin($admin)->flags($flags);
    }

    public function flags(\Illuminate\Support\Collection $flags): self
    {
        $this->flags = $flags;

        return $this;
    }

    public function viewData(): array
    {
        $name = $this->getAdminName();

        return [
```

```

        'id' => Str::random(),
        'time' => $this->getTime(),
        'title' => $this->getTitle($name),
        'admin' => $name,
        'changes' => $this->getChanges(),
    ];
}

protected function getTitle($adminName): string
{
    return $adminName ? "{$adminName} edited this order's flags" : "This order's flags
were edited";
}

protected function getChanges()
{
    return $this->flags->groupBy('state');
}
}

```

And a view to handle rendering the activity:

```

<div class="w-full [ aero-accordion ]">
    <input type="checkbox" id="{{ $id }}Open" aria-hidden="true">

    <div class="flex items-center">
        <label class="aero-accordion-label aero-accordion-label--inline justify-start flex-1"
for="{{ $id }}Open" aria-hidden="true">
            <div class="flex items-center">
                <div class="flex-1">
                    <div class="timeline-title">
                        {{ $title }}
                    </div>
                </div>
            </div>
        </label>

        @include('admin::activity.partials.time', compact('time'))
    </div>

```

```

<div class="visually-hidden">
    <div>
        {{ $title }}
    </div>
</div>

<div class="[ aero-accordion-content ]">
    <div class="pl-6">
        @foreach($changes as $state => $stateChanges)
            <div class="py-2">
                <div class="font-medium mb-2">{{ ucfirst($state) }}</div>

                @foreach($stateChanges as $change)
                    <div class="flex items-center mt-1">
                        &bull;&nbsp;
                        @include('admin::icons.flag', [
                            'color' => $change->flag->color
                        ])
                        @if($state === \Aero\Admin\Models\FlagHistory::ADDED)
                            <span>&nbsp;{{ $change->flag->name }} @if($change-
>new_value)({{ $change->new_value }})@endif</span>
                        @elseif($state === \Aero\Admin\Models\FlagHistory::UPDATED)
                            <span>&nbsp;{{ $change->flag->name }} ({{ $change->old_value
?? view('admin::resource-lists.placeholder') }} => {{ $change->new_value }})</span>
                        @else
                            <span>&nbsp;{{ $change->flag->name }} @if($change-
>old_value)({{ $change->old_value }})@endif</span>
                        @endif
                    </div>
                @endforeach
            </div>
        @endforeach
    </div>
</div>
</div>

```

Finally, you must extend `\Aero\Cart\Models\Order` to add this activity:

```

\Aero\Cart\Models\Order::extendActivity(function (\Aero\Cart\Models\Order $order) {
    $order->loadMissing('flagHistoryGroups.admin', 'flagHistoryGroups.histories.flag')-

```

```
>flagHistoryGroups->each(function (\Aero\Admin\Models\FlagHistoryGroup $group) use ($order) {  
    $order->addActivity(\Aero\Admin\Activities\OrderFlagsEditActivity::create(  
        $group->created_at,  
        $group->admin,  
        $group->histories  
    ));  
});  
});
```

How do I remove the manufacturers name from the order items name?

By default if a product name doesn't contain the manufacturer name already, the manufacturer name is appended to the front of the product name when converting a cart item to an order item.

If you don't want the manufacturer name appended to the product name you can set the static

`Aero\Cart\Models\OrderItem::$nameContainsManufacturer` boolean to `false`:

```
\Aero\Cart\Models\OrderItem::$nameContainsManufacturer = false;
```

Shipping

How do I set a custom order validator for shipping methods?

Shipping methods can define an order validator to determine whether they are available for a particular order.

For example, you may only want to offer some shipping methods on certain days.

First we will define an availability setting for each day:

```
$days = collect([
    'sunday' => false,
    'monday' => true,
    'tuesday' => true,
    'wednesday' => true,
    'thursday' => true,
    'friday' => true,
    'saturday' => false,
]);

\Aero\Cart\Models\ShippingMethod::settings('availability', function
(\Aero\Common\Settings\ModelSettingGroup $group) use ($days) {
    $days->each(function (bool $available, string $day) use ($group) {
        $group->boolean("{ $day}_delivery")->default($available);
    });
});
```

The `setOrderValidator` method can then use these settings to determine whether the shipping method is available today:

```
\Aero\Cart\Models\ShippingMethod::setOrderValidator(function (\Aero\Cart\Models\ShippingMethod
$method) use ($days) {
```

```
$day = $days->keys()[now()->dayOfWeek];
```

```
return $method->setting("availability.{$day}_delivery");  
});
```

The validator receives the shipping method, order and order item group and should return `true` when it can be used for the order, or `false` when it should be excluded.

Parameters

Parameter	Type	Description
<code>\$method</code>	<code>\Aero\Cart\Models\ShippingMethod</code>	The shipping method being validated.
<code>\$order</code>	<code>\Aero\Cart\Models\Order</code>	The order being validated.
<code>\$orderItemGroup</code>	<code>string null</code>	The order item group being validated (<code>null</code> when not in a group).

How do I set a custom order validator for shipping rates?

Shipping rates can define an order validator to determine whether they are available for a particular order.

For example, you may only want to offer a shipping rate only for certain customer groups.

First, define an availability setting to allow this:

```
\Aero\Cart\Models\ShippingRate::settings('availability', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->boolean('restricted_by_customer_group')->default(false);

    $group->eloquent('allowed_customer_groups', \Aero\Account\Models\CustomerGroup::class)-
>multiple()->dontLoadModels()
    ->showIf('settings.availability.restricted_by_customer_group', true);

    $group->eloquent('disallowed_customer_groups', \Aero\Account\Models\CustomerGroup::class)-
>multiple()->dontLoadModels()
    ->showIf('settings.availability.restricted_by_customer_group', true);
});
```

The `setOrderValidator` method can then use these settings to determine whether the shipping rate is available for a customer:

```
\Aero\Cart\Models\ShippingRate::setOrderValidator(function (\Aero\Cart\Models\ShippingRate
$rate) {
    if (! $rate->settings('availability.restricted_by_customer_group')) {
        return true;
    }

    $disallowedGroups = $rate->settings('availability.disallowed_customer_groups') ?? [];
    $allowedGroups = $rate->settings('availability.allowed_customer_groups') ?? [];
```

```

    $customerGroupId = optional($rate->shippingMethod->order->customer)->customer_group_id ??
    null;

    if (in_array($customerGroupId, $disallowedGroups)) {
        return false;
    }

    if (empty($allowedGroups)) {
        return true;
    }

    return in_array($customerGroupId, $allowedGroups);
});

```

The validator receives the shipping rate, order and order item group and should return `true` when it can be used for the order, or `false` when it should be excluded.

Parameters

Parameter	Type	Description
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being validated.
<code>\$order</code>	<code>\Aero\Cart\Models\Order</code>	The order being validated.
<code>\$items</code>	<code>\Aero\Common\Eloquent\JavaScriptableCollection</code>	The items of the order, or the order item group.

How do I set a custom order validator for shipping rates (for subscriptions)?

Shipping rates can define a subscription order validator to determine whether they are available for a particular subscription order.

For example, you may want to make a shipping rate available for one-time orders but explicitly opt it into subscription orders.

First, define an availability setting:

```
\Aero\Cart\Models\ShippingRate::settings('availability', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->boolean('for_subscription_orders')->default(false);
});
```

The `setSubscriptionOrderValidator` method can then use this setting to determine whether the shipping rate is available for subscription orders:

```
\Aero\Cart\Models\ShippingRate::setSubscriptionOrderValidator(function
(\Aero\Cart\Models\ShippingRate $rate) {
    return $rate->setting('availability.for_subscription_orders');
});
```

The validator receives the shipping rate, order and subscription plan and should return `true` when it can be used for the order, or `false` when it should be excluded.

Parameters

Parameter	Type	Description
-----------	------	-------------

<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being validated.
<code>\$order</code>	<code>\Aero\Cart\Models\Order</code>	The order being validated.
<code>\$plan</code>	<code>\Aero\Subscription\Models\SubscriptionPlan</code>	The subscription plan being validated.

How do I set a custom price totalizer for shipping rates?

Shipping rates can define a custom price totalizer to adjust the calculated shipping price before it is returned to the customer.

For example, you may want to add a fixed cost for each item in the order.

First, define the cost per item on the shipping method:

```
\Aero\Cart\Models\ShippingMethod::settings('extras', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->float('cost_per_item');
});
```

The `setPriceTotalizer` method can then add the per-item cost to the shipping rate's calculated price:

```
\Aero\Cart\Models\ShippingRate::setPriceTotalizer(function ($price,
\Aero\Cart\Models\ShippingRate $rate) {
    $costPerItem = $rate->setting('extras.cost_per_item')
        ?? $rate->shippingMethod->setting('extras.cost_per_item');

    if (! $costPerItem) {
        return null;
    }

    $itemAmount = $rate->order->items->sum('quantity');

    return $price + ($itemAmount * $costPerItem * 100);
});
```

The totalizer receives the price and the shipping rate as parameters and should return the total in pence or `null`.

Returning `null` means that the default price totalizer will be used.

Parameters

Parameter	Type	Description
<code>\$price</code>	<code>float</code>	The total price in pence.
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being evaluated.

How do I set a custom applicable items filter for shipping rates?

Shipping rates can define a custom-applicable items filter to control which order items are considered when determining whether a rate is applicable.

For example, you may want to restrict a shipping method to products from specific manufacturers.

First, define settings for enabling manufacturer restrictions and selecting the allowed and disallowed manufacturers:

```
\Aero\Cart\Models\ShippingMethod::settings('availability', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->boolean('restricted_by_manufacturer')->default(false);

    $group->eloquent('allowed_manufacturers', \Aero\Catalog\Models\Manufacturer::class)-
>multiple()->dontLoadModels()
    ->showIf('settings.availability.restricted_by_manufacturer', true);

    $group->eloquent('disallowed_manufacturers', \Aero\Catalog\Models\Manufacturer::class)-
>multiple()->dontLoadModels()
    ->showIf('settings.availability.restricted_by_manufacturer', true);
});
```

The `setApplicableItemsFilter` method can then filter out items that don't meet the manufacturer restrictions:

```
\Aero\Cart\Models\ShippingRate::setApplicableItemsFilter(function ($items,
\Aero\Cart\Models\ShippingRate $rate) {
    if (! $rate->shippingMethod->setting('availability.restricted_by_manufacturer')) {
        return $items;
    }
});
```

```

        $allowedManufacturerIds = $rate->shippingMethod-
>setting('availability.allowed_manufacturers');

        $disallowedManufacturerIds = $rate->shippingMethod-
>setting('availability.disallowed_manufacturers');

        $hasDisallowedItem = $items->some(function (\Aero\Cart\Models\OrderItem $item) use
($allowedManufacturerIds, $disallowedManufacturerIds) {

            if (in_array($item->buyable->manufacturer_id, $disallowedManufacturerIds)) {
                return true;
            }

            if (empty($allowedManufacturerIds)) {
                return false;
            }

            return ! in_array($item->buyable->manufacturer_id, $allowedManufacturerIds);
        }

        if ($hasDisallowedItem) {
            return collect();
        }

        return $items;
    });

```

The filter receives the order price and the shipping rate as parameters and should return a collection of applicable items.

Parameters

Parameter	Type	Description
<code>\$items</code>	<code>\Aero\Common\Eloquent\JavaScriptableCollection</code>	The order items being evaluated.
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being evaluated.

How do I set a custom item weight totalizer for shipping rates?

Shipping rates can define a custom item weight totalizer to control how item weights are calculated when determining whether a rate is applicable.

For example, you may want certain product variants to be exempt from the shipping weight calculation.

First, add a setting to the variant:

```
\Aero\Catalog\Models\Variant::settings('shipping', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->boolean('is_weight_exempt')->default(false);
});
```

The `setItemWeightTotalizer` method can then exclude weight-exempt variants when calculating the total weight:

```
\Aero\Cart\Models\ShippingRate::setItemWeightTotalizer(function ($items) {
    return $items
        ->reject(function (\Aero\Cart\Models\OrderItem $item) {
            return $item->buyable->setting('shipping.is_weight_exempt');
        })
        ->sum(function (\Aero\Cart\Models\OrderItem $item) {
            return $item->buyable->weight;
        });
});
```

The totalizer receives the order items and the shipping rate as parameters and should return the total weight of the items or `null`.

Returning `null` means the default item weight totalizer will be used.

Parameters

Parameter	Type	Description
<code>\$items</code>	<code>\Aero\Common\Eloquent\JavaScriptableCollection</code>	The order items being evaluated.
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being evaluated.

How do I set a custom item volume totalizer for shipping rates?

Shipping rates can define a custom item volume totalizer to control how item volumes are calculated when determining whether a rate is applicable.

For example, you may want to account for additional packaging space by applying a percentage allowance to the volume of each item.

First, define the packaging allowance on the shipping method:

```
\Aero\Cart\Models\ShippingMethod::settings('extras', function
(\Aero\Common\Settings\ModelSettingGroup $group) {
    $group->integer('packaging_allowance')->default(20);
});
```

The `setItemVolumeTotalizer` method can then apply the allowance when calculating the total volume:

```
\Aero\Cart\Models\ShippingRate::setItemVolumeTotalizer(function ($items,
\Aero\Cart\Models\ShippingRate $rate) {
    $packagingAllowance = 1 + ($rate->shippingMethod->setting('extras.packaging_allowance') /
100);

    return $items->sum(function (\Aero\Cart\Models\OrderItem $item) use ($packagingAllowance)
    {
        return $item->buyable->volume * $packagingAllowance;
    });
});
```

The totalizer receives the order items and the shipping rate as parameters and should return the total volume of the items or `null`.

Returning `null` means the default item volume totalizer will be used.

Parameters

Parameter	Type	Description
<code>\$items</code>	<code>\Aero\Common\Eloquent\JavaScriptableCollection</code>	The order items being evaluated.
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being evaluated.

How do I set a custom item quantity totalizer for shipping rates?

Shipping rates can define a custom item quantity totalizer to control how item quantities are calculated when determining whether a rate is applicable.

For example, you may want sample products to be excluded from the item quantity used when evaluating quantity-based shipping rules.

First, define a setting on the variant:

```
\Aero\Catalog\Models\Variant::settings(function (\Aero\Common\Settings\ModelSettingGroup $group) {  
    $group->boolean('is_sample')->default(false);  
});
```

The `setItemQuantityTotalizer` method can then exclude sample products when calculating the total quantity:

```
\Aero\Cart\Models\ShippingRate::setItemQuantityTotalizer(function ($items) {  
    return $items  
        ->reject(function (\Aero\Cart\Models\OrderItem $item) {  
            return $item->buyable->setting('is_sample');  
        })  
        ->sum('quantity');  
});
```

The totalizer receives the order items and the shipping rate as parameters and should return the total quantity of the items or `null`.

Returning `null` means that the default item quantity totalizer will be used.

Parameters

Parameter	Type	Description
<code>\$items</code>	<code>\Aero\Common\Eloquent\JavaScriptableCollection</code>	The order items being evaluated.
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being evaluated.

How do I set a custom item price totalizer for shipping rates?

Shipping rates can define a custom item price totalizer to control how the total item price is calculated when determining whether a rate is applicable.

For example, you may want to calculate the shipping rate's price threshold using the original item prices rather than the discounted prices.

First, add settings to control whether discounts should be excluded:

```
\Aero\Cart\Models\ShippingRate::settings('extras', function
(\Aero\Settings\Models\ModelSettingGroup $group) {
    $group->boolean('exclude_discount_from_total');
});
```

The `setItemPriceTotalizer` method can then calculate the total using the items' original prices when both the shipping rate has this option enabled:

```
\Aero\Cart\Models\ShippingRate::setItemPriceTotalizer(function ($items,
\Aero\Cart\Models\ShippingRate $rate) {
    if (! $rate->setting('extras.exclude_discount_from_total')) {
        return null;
    }

    $total = $items->sum(function (\Aero\Cart\Models\OrderItem $item) {
        return $item->subtotal_price->inc()->value;
    });

    return round($total);
});
```

The totalizer receives the order items and the shipping rate as parameters and should return the total in pence or `null`

Returning `null` means the default item price totalizer will be used.

Parameters

Parameter	Type	Description
<code>\$items</code>	<code>\Aero\Common\Eloquent\JavaScriptableCollection</code>	The order items being evaluated.
<code>\$rate</code>	<code>\Aero\Cart\Models\ShippingRate</code>	The shipping rate being evaluated.

Subscriptions

What is the subscription order builder?

The Subscription Order Builder runs whenever a subscription is processed and requires a new order. It creates the new order based on the original order associated with the subscription.

For example, you can add a flag to every order created by a subscription.

First add a setting to the `admin` group:

```
\Aero\Common\Facades\Settings::group('acme', function (\Aero\Common\Settings\SettingGroup $group) {  
    $group->eloquent('subscription_flag', \Aero\Common\Models\Flag::class)->dontLoadModel()-  
>section('Orders');  
});
```

The `SubscriptionOrderBuilder` can then use this setting to attach the flag to the new order:

```
\Aero\Subscription\Pipelines\SubscriptionOrderBuilder::extend(function  
(\Aero\Subscription\Pipelines\SubscriptionOrderBuilder $builder, \Closure $next) {  
    $flagId = setting('acme.subscription_flag');  
  
    if (! is_int($flagId)) {  
        return $next($builder);  
    }  
  
    $builder->order->flags()->attach($flagId);  
  
    return $next($builder);  
});
```

Parameters

Parameter	Type	Description
<code>\$builder</code>	<code>\Aero\Subscription\Pipelines\SubscriptionOrderBuilder</code>	The subscription order builder.
<code>\$next</code>	<code>\Closure</code>	The next step in the order item builder.

Attributes

Attribute	Type	Description
<code>\$subscription</code>	<code>\Aero\Subscription\Models\Subscription</code>	The subscription being processed.
<code>\$originalOrder</code>	<code>\Aero\Cart\Models\Order</code>	The original order associated with the subscription.
<code>\$order</code>	<code>\Aero\Cart\Models\Order</code>	The new order being created for the subscription