

Discount Management

- [Discounts](#)
 - [How do I add a custom amount resolver?](#)
 - [How can I intercept discount code handling?](#)
 - [How can I apply multiple discount types using one code?](#)
- [Vouchers](#)
 - [How do I add to the voucher activity timeline?](#)
- [Price Lists](#)
 - [How do I resolve custom price lists?](#)

Discounts

How do I add a custom amount resolver?

The cart discount implementations use the `\Aero\Cart\Traits\DeterminesAmount` trait, which allows you to set a custom amount resolver for a discount type

For example you can set a limit for percentage discounts:

```
\Aero\Cart\PercentageDiscount::resolveAmount(function (\Aero\Cart\CartDiscount $discount,
float $percentage, $items) {
    // Limit in pence, e.g. 100 = £1
    $limit = $discount->model->additional('percentage_discount_limit');

    if (! $limit) {
        return $percentage;
    }

    $itemsTotal = $items->sum(function (\Aero\Cart\CartItem $item) use ($percentage) {
        return $item->total()->inc()->value;
    });

    $discountTotal = $itemsTotal * ($percentage / 100);

    return $discountTotal > $limit
        ? ($limit / $itemsTotal) * 100
        : $percentage;
});
```

By default if multiple amounts are resolved for a discount by resolvers (e.g. if you had another custom resolver for percentage discounts) the largest amount is resolved. This can be changed to instead prioritise the smallest amount:

```
\Aero\Cart\PercentageDiscount::giveMinResolvedAmount();
```

In fact, you can even take full control over which amount would be resolved. For example, you could resolve the largest amount for customers but the lowest amount for guests:

```
\Aero\Cart\PercentageDiscount::determineResolvedAmount(function(array $amounts) {  
    $customer = \Aero\Cart\Facades\Cart::customer();  
  
    return $customer ? max($amounts) : min($amounts);  
});
```

How can I intercept discount code handling?

You can use the `addCodeHandler()` method to add a discount handler to the cart, it will run the provided callback whenever a code is added to the cart:

```
\Aero\Cart\Cart::addCodeHandler(function (\Aero\Cart\Cart $cart, string $code) {  
    if ($code === 'acme') {  
        // Do external stuff...  
  
        return true;  
    }  
  
    return false;  
});
```

Return `true` to handle the code yourself. Return `false` to let the Cart handle it normally.

How can I apply multiple discount types using one code?

There are two `aerocargo` modules that can be used to achieve this behaviour:

[aerocargo/discount-groups](#)

This module allows you to create a discount group with a code and specify the discounts you want to be applied when that code is applied to the cart.

So you can create a group that adds a 20% off discount & a free shipping discount, while maintaining the flexibility of both being able to be added separately.

[aerocargo/connected-discounts](#)

This module allows you to connect a discount to another one via the discount edit page. Then when a code that has any connected discounts is applied, all connected discounts will also be applied.

So you can connect a free shipping discount to a 20% off discount, meaning when the 20% off discount is added so is free shipping.

Unless you also connect the 20% off discount to the free shipping as well, it wont work both ways.

Vouchers

How do I add to the voucher activity timeline?

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

`\Aero\Admin\Activity`:

```
class CommentActivity extends \Aero\Admin\Activities\Activity
{
    protected $comment;

    protected $view = 'admin::activity.comment';

    protected $key = 'comment-activity';

    protected $name = 'Comments';

    public static function create(\Carbon\Carbon $time, \Aero\Admin\Models\Comment $comment,
    ?\Aero\Admin\Models\Admin $admin = null): self
    {
        return (new self($time))->admin($admin)->comment($comment);
    }

    public function comment(\Aero\Admin\Models\Comment $comment): self
    {
        $this->comment = $comment;

        return $this;
    }

    public function viewData(): array
    {
        return [
            'time' => $this->getTime(),
            'admin' => $this->admin,
```

```

        'adminName' => $this->getAdminName(),
        'comment' => $this->comment,
    ];
}

protected function adminNameResolverData(): array
{
    return array_merge(parent::adminNameResolverData(), [
        'comment' => $this->comment,
    ]);
}
}

```

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>

```

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

```

\Aero\Cart\Models\Voucher::extendActivity(function (\Aero\Cart\Models\Voucher $voucher) {
    $voucher->loadMissing('comments.admin');

    $voucher->comments->each(function (Comment $comment) use ($voucher) {
        $voucher->addActivity(\Aero\Admin\Activities\CommentActivity::create(
            $comment->created_at,
            $comment,
            $comment->admin
        ));
    });

```

```
});
```

```
});
```

Price Lists

How do I resolve custom price lists?

To resolve custom price lists you can create a customer group in the Admin and create price lists that are only available for customers in that group.

First, create a setting for this customer group so that you can easily access it in code:

```
\Aero\Common\Facades\Settings::group('acme', function (\Aero\Common\Settings\SettingGroup $group) {
    $group->eloquent('customer_group', \Aero\Account\Models\CustomerGroup::class)->dontLoadModel();
});
```

Then add the logic to conditionally resolve price lists from this group for a customer instead of their own:

```
\Aero\Common\Models\PriceList::addCustomerGroupResolver(function () {
    if (! session()->has('newsletter_subscriber')) {
        return null;
    }

    // The customer group id
    return setting('acme.customer_group');
});
```

Parameters

Parameter	Type	Description
<code>\$customer</code>	<code>?\Aero\Account\Models\Customer</code>	The currently authenticated customer.