

Catalog Management

- [Products](#)
 - [How do I remove the manufacturer from the product slug/url?](#)
- [Listing Rules](#)
 - [What is the listing rules context builder?](#)
 - [What is the listing rules data formatter pipeline?](#)
 - [How can I add a custom listing rule?](#)

Products

How do I remove the manufacturer from the product slug/url?

The manufacturer name is automatically prepended to the product slug (its URL). If you wish for this to not happen you can set the static `Aero\Catalog\Models\Product::$slugContainsManufacturer` boolean to `false`:

```
\Aero\Catalog\Models\Product::$slugContainsManufacturer = false;
```

Listing Rules

What is the listing rules context builder?

The Listing Rules Context Builder is used to build the rule-specific context that listing rules need when being displayed in the Admin UI.

For example, the `\Aero\Admin\Pipelines\Pipes\AttachTaggedWithRuleContext` pipe attaches the hydrated tags data for the `\Aero\Catalog\Rules\Types\TaggedWithRule` rule:

```
class AttachTaggedWithRuleContext
{
    public function handle(\Aero\Admin\Pipelines\ListingRuleContextBuilder $pipeline,
\Closure $next)
    {
        $tagIds = $pipeline->rules
            ->where('type', \Aero\Catalog\Rules\Types\TaggedWithRule::KEY)
            ->flatMap(function (ListingRule $rule) {
                return collect($rule->options['filters'] ?? [])->flatten()->toArray();
            })
            ->unique();

        $pipeline->data[\Aero\Catalog\Rules\Types\TaggedWithRule::KEY] = [
            'tags' => $tagIds->isEmpty()
                ? \Aero\Catalog\Models\Tag::query()
                    ->select('id', 'tag_group_id', 'name')
                    ->with('group')
                    ->whereIn('id', $tagIds)->get()
                    ->mapWithKeys(function (\Aero\Catalog\Models\Tag $tag) {
                        $id = $tag->getKey();

                        return [
                            $id => [
                                'value' => (string) $id,
                                'name' => $tag->getAttribute('name'),
```

```

        'group_id' => (string) $tag->getRelation('group')->getKey(),
        'group' => $tag->getRelation('group')->getAttribute('name'),
    ],
    ];
    });
    ->toArray()
    : [],
    ];

    return $next($pipeline);
}
}

```

Parameters

Parameter	Type	Description
<code>\$pipeline</code>	<code>\Aero\Admin\Pipelines\ListingRulesContextBuilder</code>	The listing rules context builder.
<code>\$next</code>	<code>\Closure</code>	The next step in the builder.

Attributes

Attribute	Type	Description
<code>\$rules</code>	<code>\Illuminate\Support\Collection</code>	All the rules corresponding to the model (that is being viewed in the Admin).
<code>\$data</code>	<code>array</code>	The array of data the rules need (for display purposes only).

What is the listing rules data formatter pipeline?

The Listing Rules Data Formatter is used to format the listing rules data into an array for the Admin UI page.

For example, the `\Aero\Admin\Pipelines\Pipes\AttachListingRuleAttributes` attaches the attributes to the array:

```
class AttachListingRuleAttributes
{
    public function handle(\Aero\Admin\Pipelines\ListingRulesDataFormatter $pipeline, Closure $next)
    {
        $pipeline->data['id'] = $pipeline->rule->getKey();

        $pipeline->data['type'] = $pipeline->rule->getAttribute('type');

        $pipeline->data['rule_logic'] = $pipeline->rule->getAttribute('rule_logic') ?? 'must';

        $pipeline->data['logic'] = $pipeline->rule->getAttribute('logic') ?? 'and';

        $pipeline->data['group'] = $pipeline->rule->getAttribute('group') ?? 0;

        $pipeline->data['options'] = $pipeline->rule->getAttribute('options') ?? [];

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

The formatter also assigns the context built by the `ListingRulesContextBuilder` as view data to the rule.

Parameters

Parameter	Type	Description
<code>\$pipeline</code>	<code>\Aero\Admin\Pipelines\ListingRulesDataFormatter</code>	The listing rules data formatter.
<code>\$next</code>	<code>\Closure</code>	The next step in the formatter.

Attributes

Attribute	Type	Description
<code>\$rule</code>	<code>\Aero\Catalog\Models\ListingRule</code>	The listing rule being formatted.
<code>\$data</code>	<code>array</code>	The array of formatted data for the Admin UI.

How can I add a custom listing rule?

To add a custom listing rule you need to create a class that extends

```
Aero\Catalog\Rules\ListingRule\AttributedWithRule
```

.

For example we will add a rule that checks if the listing has specific attributes.

First, we need to index the attribute ids so that they can be queried easily:

```
\Aero\Search\Elastic\Documents\ListingDocument::add('search-result', function
(\Aero\Search\Elastic\Documents\ListingDocument $document) {
    return [
        'attribute_ids' => $document->getModel()->attributes->map(function
(\Aero\Catalog\Models\Attribute $attribute) {
            return (string) $attribute->getKey();
        }),
    ];
});
```

Now, we can make the rule:

```
class AttributedWithRule extends \Aero\Catalog\Rules\ListingRule
{
    public const KEY = 'attributes';

    public string $name = 'have one of these attributes';

    public string $view = 'acme::attributes';

    protected array $options = [
        'attributes' => [],
    ];

    public function viewData(array $context): array
```

```

{
    return [
        'attributes' => collect($this->options['attributes'] ?? [])
            ->flatten()
            ->map(function ($id) use ($context) {
                return $context['attributes'][$id] ?? null;
            })
            ->toArray(),
    ];
}

public function apply(\Aero\Search\Contracts\SearchBuilder $builder):
?\Elastica\Query\BoolQuery
{
    $attributeIds = $this->options['attributes'] ?? [];

    if (! is_array($attributeIds)) {
        return null;
    }

    return (new BoolQuery())->addMust(
        new \Elastica\Query\Terms('search-result.attribute_ids', $attributeIds),
    );
}

public function optionRules(): array
{
    return [
        'attributes' => 'required|array',
        'attributes.*' => 'required|distinct|int|exists:attributes,id',
    ];
}
}

```

For the UI to work we must build the context for our rule:

```

\Aero\Admin\Pipelines\ListingRulesContextBuilder::extend(function
(\Aero\Admin\Pipelines\ListingRulesContextBuilder $pipeline, \Closure $next) {
    $attributeIds = $pipeline->rules
        ->where('type', \Aerocargo\Acme\AttributedWithRule::KEY)

```

```

->flatMap(function (\Aero\Catalog\Models\ListingRule $rule) {
    return collect($rule->options['attributes'] ?? [])->flatten()->toArray();
})
->unique();

$pipeline->data[\Aerocargo\Acme\AttributedWithRule::KEY] = [
    'attributes' => $attributeIds->isNotEmpty()
        ? Attribute::query()
            ->select('id', 'name')
            ->whereIn('id', $attributeIds)->get()
            ->mapWithKeys(function (\Aero\Catalog\Models\Attribute $attribute) {
                $id = $attribute->getKey();

                return [
                    $id => [
                        'value' => (string) $id,
                        'name' => $attribute->getAttribute('name'),
                    ]
                ];
            })
            ->toArray()
        : [],
];

return $next($pipeline);
});

```

And add the view:

```

<template v-for="attribute in builder.rule.view_data.attributes ?? []">
    <input type="hidden" :name="{{ $name }}" + '[attributes][]' :value="attribute.value">
</template>

<label :for="{{ $id }}" + '-attributes-option">Attributes</label>
<searchable-select :id="{{ $id }}" + '-attributes-option' class="mt-2 w-full"
    :input-name="'rules[' + builder.group_index + '][' + builder.rule_index +
    '][view_data][attributes]'"
    url="{{ route('admin.catalog.attributes.search') }}" track-by="value"
    label="name" :multiple="true"
    v-model="builder.rule.view_data.attributes"></searchable-select>

```

