

Search

- [General](#)
 - [How do I re-index documents?](#)
 - [How do I re-build documents?](#)
- [Documents](#)
 - [What are the available documents?](#)
 - [How can I extend the document structure?](#)
- [Listings](#)
 - [What is the listing attributes pipeline?](#)
 - [How can I add custom attributes to the listing model?](#)
 - [How can I add a custom facet?](#)
 - [How can I pre-filter listings?](#)
 - [How do I create a custom sort order for the listings pages?](#)
- [Tutorials](#)
 - [How can I distribute search jobs across multiple queues?](#)

General

How do I re-index documents?

Aero automatically manages indexing changes to documents.

For example, if a product's stock level changes the document is updated to reflect the current version.

Adding and removing documents is also taken care of for you, sparing the need to schedule in daily tasks to update the store's entire catalog. However, in cases where a large amount of data has been manually changed, or an addition to the document structure has been made, a re-index of all documents can be carried out using the following command:

```
php artisan aero:search:reindex
```

If there have only been modifications to a certain document (for example, only the listing document structure has been extended) then the document `type` can be passed as an option, which will only re-index the documents with the provided type.

The available types correspond to the names of the elastic documents, `listing`, and `product`:

```
// Only listings
php artisan aero:search:reindex --type=listing

// Only products
php artisan aero:search:reindex --type=product
```

How do I re-build documents?

At times (especially during the development phase of a store build), it may be necessary to delete and documents and freshly index them. This process can be achieved using the following command:

```
php artisan aero:search:rebuild
```

Due to the nature of how this process runs, there will be a temporary period where there are no listings or products shown.

This can be circumvented by using the `--swap` option:

```
php artisan aero:search:rebuild --swap
```

This will aliases to ensure there is no downtime, it works by creating a new alias, rebuilding the data for that alias and only after it is complete swapping over to that alias. This means there will be no period where there are no listings or products shown.

Documents

What are the available documents?

There are 2 documents available:

- `Aero\Search\Elastic\Documents\ListingsDocument` used to hold information about the listings that are shown on the storefront.
- `Aero\Search\Elastic\Documents\ProductDocument` used to hold information about the products within the store's catalog. It is generally used in the backend admin, to allow for quick access when managing products.

How can I extend the document structure?

To extend a document, use the `add()` method, to define a closure that will return an `array` of data to merge into the document structure:

```
\Aero\Search\Elastic\Documents\ListingDocument::add('search-result', function ($document) {  
    $colors = $document->getModel()  
        ->visibleVariants()  
        ->pluck('attributes')  
        ->flatten()  
        ->filter(function ($attribute) {  
            return strpos(strtolower($attribute->group->name), 'color') !== false;  
        })  
        ->pluck('name')  
        ->unique()  
        ->values()  
        ->all();  
  
    return [  
        'colors' => $colors,  
    ];  
});
```

Listings

What is the listing attributes pipeline?

The listing attributes pipeline is used to build the attributes for the `Aero\Search\Elastic\Models\Listing` model.

It can be extended to add new attributes or modify existing attributes.

For example you can modify the listing's name to include the attributes when the listing is split by them.

```
\Aero\Search\Elastic\Pipelines\ListingAttributes::extend(function (array $attributes, \Closure $next) {  
    $splitAttributeGroups = collect($attributes['attribute_groups'] ?? [])  
        ->filter(function ($attributeGroup) {  
            return $attributeGroup['split'] ?? false;  
        });  
  
    if ($splitAttributeGroups->isEmpty()) {  
        return $next($attributes);  
    }  
  
    $attributeNames = $splitAttributeGroups->flatMap(function (array $attributeGroup) {  
        return collect($attributeGroup['attributes'] ?? [])->pluck('name');  
    });  
  
    $attributes['name'] = "{$attributeNames->implode(' ')} {$attributes['name']}";  
  
    return $next($attributes);  
});
```

This means if you have a product split by a `Colour` attribute group, each listing would now be named `{colour} {name}`, e.g. Red Puffer Jack.

Parameters

Parameter	Type	Description
<code>\$attributes</code>	<code>array</code>	The attributes for the listing model.
<code>\$next</code>	<code>\Closure</code>	The next step in the pipeline.

How can I add custom attributes to the listing model?

You can extend the `\Aero\Search\Elastic\Pipelines\ListingAttributes` pipeline to add custom attributes to the listing model.

For example you can modify the listing's name to include the attributes when the listing is split by them.

```
\Aero\Search\Elastic\Pipelines\ListingAttributes::extend(function (array $attributes, \Closure $next) {  
    $splitAttributeGroups = collect($attributes['attribute_groups'] ?? [])  
        ->filter(function ($attributeGroup) {  
            return $attributeGroup['split'] ?? false;  
        });  
  
    if ($splitAttributeGroups->isEmpty()) {  
        return $next($attributes);  
    }  
  
    $attributeNames = $splitAttributeGroups->flatMap(function (array $attributeGroup) {  
        return collect($attributeGroup['attributes'] ?? [])->pluck('name');  
    });  
  
    $attributes['name'] = "{$attributeNames->implode(' ')} {$attributes['name']}";  
  
    return $next($attributes);  
});
```

This means if you have a product split by a `Colour` attribute group, each listing would now be named `{colour} {name}`, e.g. Red Puffer Jack.

How can I add a custom facet?

You can add a custom facet by extending the listing document.

For example you can add a Stock Status facet:

```
\Aero\Search\Elastic\Documents\ListingDocument::add('string-facets', function ($document) {  
    $stock = $document->getModel()->availableStock() > 0;  
  
    return [  
        \Aero\Search\Elastic\Facet::create('stock', 'Stock Status', (int) $stock, $stock ? 'In  
stock' : 'Out of stock'),  
    ];  
});
```

You must run `php artisan aero:search:reindex --type=listing` for these document changes to be applied to existing listings.

And then register it so that it appears on the listings page as a facet:

```
\Aero\Search\FacetSettings::registerFacet('stock', 'Stock Status');
```

How can I pre-filter listings?

There may be occasions where you need to pre-filter listings. For example, you may want to only show listings that have stock:

```
\Aero\Search\Elastic\Repositories\ElasticListings::extend(function
(\Aero\Search\Elastic\Repositories\ElasticListings $listings) {
    $filter = new \Elastica\Query\Term(['number-sort.has-stock' => ['value' => 1]]);

    $listings->base()->getQuery()->addFilter($filter);
});
```

Parameters

Parameter	Type	Description
<code>\$listings</code>	<code>\Aero\Search\Elastic\Repositories\ElasticListings</code>	The elastic listing repository.

Methods

Method	Description
<code>base()</code>	Returns the underlying <code>\Elastica\Query</code> instance, use <code>->getQuery()</code> to access the <code>\Elastica\Query\BoolQuery</code> directly.

How do I create a custom sort order for the listings pages?

When you set the sort get parameter in a listing pages url (e.g. ?sort=name-az), a method is called on the `\Aero\Search\Elastic\Query\Builder` class (for name-az the method `sortByNameAz` is called).

You can find the default sort by methods in the `Aero\Search\Elastic\Concerns\CanBeSorted` trait.

To add a custom sort to the listings page you therefore just need to macro a method to the `\Aero\Search\Elastic\Query\Builder` class using the correct naming convention.

For example, we will add a summary sort:

```
\Aero\Search\Elastic\Documents\ListingDocument::add('string-sort', function
(\Aero\Search\Elastic\Documents\ListingDocument $document) {
    return [
        'summary' => $document->getModel()->product->getTranslation('summary', $document-
>getLanguage()),
    ];
});

\Aero\Search\Elastic\Query\Builder::macro('sortBySummaryAz', function () {
    $this->query->addSort([
        'string-sort.summary' => [
            'order' => 'asc',
        ],
    ]);
});

\Aero\Search\Elastic\Query\Builder::macro('sortBySummaryZa', function () {
    $this->query->addSort([
        'string-sort.summary' => [
```

```

        'order' => 'desc',
    ],
    []);
});

```

You must run `php artisan aero:search:reindex --type=listing` for these document changes to be applied to existing listings.

You can add these as selectable options in the sort-by dropdown on the listings page via the `search` settings `sort_by_options`.

Advanced Query

It's possible to do a more advanced query for your Elasticsearch sort by. You can make use of [Elasticsearch's Painless scripting language](#)

For example, you can add a sort that will execute Java to randomly sort listings:

```

\Aero\Search\Elastic\Query\Builder::macro('sortByRand', function () {
    $this->query->addSort([
        '_script' => [
            'type' => 'number',
            'script' => [
                'lang' => \Elastica\Script\Script::LANG_PAINLESS,
                'source' => $this->loadJavaFile(__DIR__.'../../scripts/random-sort.java'),
                'params' => $this->parameters,
            ],
            'order' => 'asc',
        ],
    ],
    []);
});

```

```

// random-sort.java

def sorts = new ArrayList();

Random rand = new Random();
sorts.add(rand.nextInt(1000000));

```

```
if (sorts.size() == 0) {  
    sorts.add(0);  
}
```

```
Collections.min(sorts.stream().map(Double::doubleToRawLongBits).collect(Collectors.toList()));
```

Tutorials

How can I distribute search jobs across multiple queues?

You can use the `setQueueResolver()` method to distribute search jobs across multiple queues:

For example, let's say you have 10 search queues you want to utilise, you could distribute search jobs between them randomly:

```
\Aero\Search\Jobs\SearchJob::setQueueResolver(function () {  
    return 'search-' . rand(1, 10);  
});
```

Or equally:

```
protected static $queueIndex = 0;  
  
public function boot()  
{  
    \Aero\Search\Jobs\SearchJob::setQueueResolver(function () {  
        return 'search-' . (self::$queueIndex++ % 10 + 1);  
    });  
}
```

The examples will dispatch the search jobs on queues from `search-1` to `search-10`.