

Resource Lists

Resource Lists

- [How do I create a custom bulk action?](#)
- [What are the default resource lists?](#)
- [How do I create my own resource list?](#)
- [How do I extend an existing resource list?](#)
- [What is a bulk action?](#)
- [How do I add a custom column to the product list?](#)
- [How do I add a custom column to the order list?](#)

How do I create a custom bulk action?

If we wish to add a new bulk action to a screen within the admin, all we have to do is create the bulk action using the **BulkAction** facade. The bulk action also requires a specific **Resource List**, for example for **Products**.

```
BulkAction::create(BulkActionClass::class, ResourceList::class)
    ->permissions('bulkaction.permission')
    ->title('Title');
```

The above code would be instantiated within a **Service Provider**, whether it is a module service provider or the **AppServiceProvider**.

Example:

If we were to create a new bulk action for exporting products, we would first need to create a class extending BulkActionJob:

```
BulkAction::create(ExportProducts::class, ProductsResourceList::class)
    ->notRunnable()
    ->permissions('products.export')
    ->title('Export products');
```

Properties

Bulk Actions have a number of properties, each responsible for affecting a different aspect of the bulk action.

Wide Content

We can choose to display the contents of our bulk action in a wider container, which can prove to be more useful for certain modules. In order to display its contents in a container stretched to the width of the screen, apply the following function to the facade while creating it:

```
BulkAction::create(BulkActionClass::class, ResourceList::class)
  ->wideContent()
  ->permissions('bulkaction.permission')
  ->title('Title');
```

Runnable & Not Runnable

Not every bulk action is the same, and it's possible to remove the **Run** button from the bulk action in order to replace it with a custom button, for example calling the API. In order to remove the built-in **Run** button for a particular bulk action, apply the following function to the facade while creating it:

```
BulkAction::create(BulkActionClass::class, ResourceList::class)
  ->notRunnable()
  ->permissions('bulkaction.permission')
  ->title('Title');
```

Confirm

We can set the bulk action to prompt the user before performing its intended action by simply chaining a method to the BulkAction creation:

```
BulkAction::create(BulkActionClass::class, ResourceList::class)
  ->confirm()
  ->permissions('bulkaction.permission')
  ->title('Title');
```

The prompt will now ask the user if they are sure of their selection.

Title

We can set a custom title/main heading for the bulk action content, this can be done by chaining a function to the facade we use for creating the bulk action, e.g.:

```
BulkAction::create(BulkActionClass::class, ResourceList::class)
  ->title('Your title')
  ->permissions('bulkaction.permission')
  ->title('Title');
```

View

Aero has the ability to display a specific view when it performs the bulk action, this can be done by chaining a function to the facade we use for creating the bulk action, e.g.:

```
BulkAction::create(BulkActionClass::class, ResourceList::class)
  ->view('namespace::bulk.form')
  ->permissions('bulkaction.permission')
  ->title('Title');
```

What are the default resource lists?

The following classes are in the **Aero\Admin\ResourceLists** namespace and any class that doesn't have a view uses the **admin::resource-lists.index** view.

Class	View
AttributeGroupsResourceList	-
BlocksResourceList	-
CategoriesResourceList	admin::catalog.categories.index
CollectionsResourceList	-
CombinationsResourceList	-
CountriesResourceList	-
CurrenciesResourceList	-
CustomerGroupsResourceList	-
CustomersResourceList	-
CustomerTaxGroupsResourceList	-
DiscountsResourceList	-
FulfillmentMethodsResourceList	-
FulfillmentsResourceList	-
MailNotificationsResourceList	-
ManufacturersResourceList	-
MissingPagesResourceList	-
OrdersResourceList	-
OrderStatusesResourceList	-
PagesResourceList	-
PaymentMethodsResourceList	-
PriceListsResourceList	-
ProductsResourceList	admin::catalog.products.index
ProductTaxGroupsResourceList	-

Class	View
RedirectsResourceList	admin::content.redirects.index
ShippingMethodsResourceList	-
SubscriptionPlansResourceList	-
SubscriptionsResourceList	-
TagGroupsResourceList	-
TaxRatesResourceList	-
TaxRulesResourceList	-
UsersResourceList	-
VouchersResourceList	-

How do I create my own resource list?

To create a resource list you need a normal Laravel controller and a class that extends **Aero\Admin\ResourceLists\AbstractResourceList**.

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;

class OrdersResourceList extends AbstractResourceList
{
    //
}
```

Your controller should be like this example but for your new resource list:

```
<?php

namespace Acme\MyModule\Http\Controllers;

use Aero\Admin\Http\Controllers\Controller;
use Aero\Admin\ResourceLists\OrdersResourceList;
use Illuminate\Http\Request;

class OrdersController extends Controller
{
    public function index(OrdersResourceList $list, Request $request)
    {
        return view('admin::resource-lists.index', [
            'list' => $list = $list(),
            'results' => $list->apply($request->all())
        ]);
    }
}
```

```

        ->with('status', 'customer', 'items', 'currency')
        ->paginate($request->input('per_page', 24) ?? 24),
    ]);
}
}

```

Adding Columns to your Resource List

To add columns to your resource list you need to implement the protected **columns()** method and make it return an array of your columns. The entries in the array should use the **Aero\Admin\ResourceLists\ResourceListColumn::create** helper method. This method expects a string parameter for the columns heading, a closure parameter for how to render the columns data (this can return a string), and an optional string parameter for the columns key (by default this is generated from the columns header).

```

<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;
use Aero\Admin\ResourceLists\ResourceListColumn;

class OrdersResourceList extends AbstractResourceList
{
    protected function columns(): array
    {
        return [
            ResourceListColumn::create('Order', function ($row) {
                return view('admin::resource-lists.link', [
                    'route' => route('admin.orders.view', array_merge(request()->all(),
['order' => $row])),
                    'text' => '#'.$row->reference,
                ]->render();
            }),
        ];
    }
}

```

Making Columns Searchable

You can use the **setSearchAction** method to define how your column is searched. This method accepts a closure (that is passed the query and search term) that defines how the search should happen and an optional string for the text to show in the search field dropdown (this is column header by default).

```
ResourceListColumn::create('Order', function ($row) {
    return view('admin::resource-lists.link', [
        'route' => route('admin.orders.view', array_merge(request()->all(), ['order' =>
$row])),
        'text' => '#'.$row->reference,
    ]->render());
})
->setSearchAction(function ($query, $term) {
    $searchTerm = ltrim($search, '#');

    $query->whereLower('reference', 'like', "%{$searchTerm}%");
}, 'Order Reference')
```

Making Columns Invisible

You can use the **invisible** method to make your column invisible. The main reason for wanting to do this is to add a search for a column that shouldn't be rendered.

```
ResourceListColumn::create('Order', function ($row) {
    return view('admin::resource-lists.link', [
        'route' => route('admin.orders.view', array_merge(request()->all(), ['order' =>
$row])),
        'text' => '#'.$row->reference,
    ]->render());
})
->invisible()
```

Handling Searching for your Resource List

When your resource list is searched without a specific column being selected a **`**handleSearch()`** function will be called to handle the search. The method will be passed the search term and you are able to use **`$this->query`** to access the query.

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;

class OrdersResourceList extends AbstractResourceList
{
    protected function handleSearch($search)
    {
        $searchTerm = ltrim($search, '#');

        $this->query->whereLower('reference', 'like', "%{$searchTerm}%");
    }
}
```

Adding Row Views to your Resource List

Row views are additional rows rendered after each complete row (the best example of this is how the order items are listed after each row of the order on the orders page). To add a row view you simply need to provide your view in the protected **`$rowViews`** array.

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;

class OrdersResourceList extends AbstractResourceList
{
    protected $rowViews = [
        'admin::resource-lists.orders.items',
    ];
}
```

Adding Buttons to your Resource List Header

Most resource list pages will require a create button. You can easily add buttons to the top of your resource list page by defining a header slot. Once you have defined a header slot you can inject your view using the admin slots helper.

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;

class OrdersResourceList extends AbstractResourceList
{
    protected $headerSlot = 'my-list.index.header.buttons';
}
```

Adding Admin Filters to your Resource List

First you will need to create an Admin Filter class as shown in the article "[How do I create a custom admin filter?](#)"

To register your newly created admin filter with your resource list you need to ensure that your admin filter class is in the resource lists protected static \$filters array.

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\Filters\Order\OrderStatusAdminFilter;
use Aero\Admin\ResourceLists\AbstractResourceList;

class OrdersResourceList extends AbstractResourceList
```

```
{
    protected $filters = [
        OrderStatusAdminFilter::class,
    ];
}
```

Adding Sort Bys to your Resource List

To add sort bys to your resource list you need to implement the protected `**sortBy()` method and make it return an array of your sort bys. The entries in the array should use the `**Aero\Admin\ResourceLists\ResourceListSortBy::create()` helper method. This method expects an array parameter holding the dropdown options that will be shown in the frontend dropdown, and a closure that will be executed when the sort option is active.

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;
use Aero\Admin\ResourceLists\ResourceListSortBy;

class OrdersResourceList extends AbstractResourceList
{
    protected function sortBy(): array
    {
        return [
            ResourceListSortBy::create([
                'order-az' => 'Order A to Z',
                'order-za' => 'Order Z to A',
            ], function ($sortBy, $query) {
                return $sortBy === 'order-az' ? $query->orderBy('reference') : $query->orderByDesc('reference');
            }),
        ];
    }
}
```

If you would like your sort by to be enabled by default then you call pass null for the first parameter like this:

```
<?php

namespace Acme\MyModule\ResourceLists;

use Aero\Admin\ResourceLists\AbstractResourceList;
use Aero\Admin\ResourceLists\ResourceListSortBy;

class OrdersResourceList extends AbstractResourceList
{
    protected function sortBy(): array
    {
        return [
            ResourceListSortBy::create(null, function ($sortBy, $query) {
                return $query->orderByRaw('coalesce(`orders`.`ordered_at`,
`orders`.`created_at`) desc');
            }),
        ];
    }
}
```

How do I extend an existing resource list?

Adding a New Column

To add columns to a resource list you can use the **addColumn**, **addColumnBefore**, or **addColumnAfter** methods from within the pipeline. These methods return a usual resource list column which allows you to chain on the other available column methods.

addColumn Method

This method accepts a column header and a closure that defines how the column will be rendered. A column added with this method will be added as the first column.

addColumnBefore Method

This method accepts the same parameters as the **addColumn** method but additionally accepts a third parameter, the key of the column this new column should be added before.

addColumnAfter Method

This method accepts the same parameters as the **addColumn** method but additionally accepts a third parameter, the key of the column this new column should be added after.

Finding the Current Column Keys

If you do not know the keys of the current columns you can use this code snippet to dump the current keys out.

```
<?php

namespace Acme\MyModule;

use Aero\Admin\ResourceLists\OrdersResourceList;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
```

```

{
    public function setup()
    {
        dd((app(OrdersResourceList::class)())->getColumns()->map->key());
    }
}

```

```

<?php

namespace Acme\MyModule;

use Aero\Admin\ResourceLists\OrdersResourceList;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        OrdersResourceList::extend(function (OrdersResourceList $list) {
            $list->addColumn('My Column', function($row) {
                return 'Example';
            });

            $list->addColumnBefore('Before Ref', function($row) {
                return $row->reference;
            }, 'order');

            $list->addColumnAfter('After Ref', function($row) {
                return $row->reference;
            }, 'order');
        });
    }
}

```

Adding a New Filter

To add a filter to a resource list you can use the **addFilter** method from within the pipeline. This method expects you to pass an admin filter class. You can create an admin filter class as shown in the article "[How do I create a custom Admin Filter?](#)"

```
<?php

namespace Acme\MyModule;

use Acme\MyModule\Filters\CreatedAtAdminFilter;
use Aero\Admin\ResourceLists\OrdersResourceList;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        OrdersResourceList::extend(function (OrdersResourceList $list) {
            $list->addFilter(CreatedAtAdminFilter::class);
        });
    }
}
```

Adding a New Sort By

To add a sort by to a resource list you can use the **addSortBy** method from within the pipeline. This method expects you to pass a **Aero\Admin\ResourceLists\ResourceListSortBy**. You can create a resource list sort by class as shown in the article "[How do I create my own Resource List?](#)"

```
<?php

namespace Acme\MyModule;

use Aero\Admin\ResourceLists\OrdersResourceList;
use Aero\Admin\ResourceLists\ResourceListSortBy;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
```

```
{
  OrdersResourceList::extend(function (OrdersResourceList $list) {
    $list->addSortBy(ResourceListSortBy::create([
      'latest' => 'Latest',
      'oldest' => 'Oldest',
    ], function ($sortBy, $query) {
      return $sortBy === 'oldest' ? $query->orderBy('created_at') : $query-
>orderByDesc('created_at');
    }));
  });
}
```

What is a bulk action?

A bulk action allows for the selection of multiple values from a list, and apply some logic to items that have only been selected. Bulk actions exist for many different **Resource Lists**, which is what is used for determining items that have been selected.

Bulk actions are usually dispatched as a job, given the bulk nature of the action, which carries out tasks in a sequential manner. There are exceptions to this rule however and Aero can also return responses straight to the browser, e.g. downloading all selected content blocks as a .zip.

How do I add a custom column to the product list?

As the products list is a resource list you can add a custom column like this:

More information on resource lists can be found in the article "[How do I extend an existing Resource List?](#)"

```
<?php

namespace Acme\MyModule;

use Aero\Admin\ResourceLists\ProductsResourceList;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        ProductsResourceList::extend(function (ProductsResourceList $list) {
            $list->addColumn('My Column', function ($row) {
                return $row->id;
            });
        });
    }
}
```

How do I add a custom column to the order list?

As the orders list is a resource list you can add a custom column like this:

More information on resource lists can be found in the article "[How do I extend an existing Resource List?](#)"

```
<?php

namespace Acme\MyModule;

use Aero\Admin\ResourceLists\OrdersResourceList;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        OrdersResourceList::extend(function (OrdersResourceList $list) {
            $list->addColumn('My Column', function ($row) {
                return '#' . $row->reference;
            });
        });
    }
}
```