

Admin Reports

Admin Reports

- [How do I create a custom report?](#)
- [How do I add lenses to my custom report?](#)
- [How do I add a table to my custom report?](#)
- [How do I add exporters to my custom report?](#)
- [How do I extend existing reports?](#)
- [How do I add admin filters to my custom report?](#)

How do I create a custom report?

To create and register a custom report you need to use the **Aero\Admin\AdminReport** facade inside of your app service providers boot method or module service providers setup method.

The facade requires a class that extends **Aero\Admin\Reports\Report**. Additionally, you can provide a title and summary (shown in the reports list in the admin) and any permissions required to use the report.

```
<?php

namespace Acme\MyModule;

use Acme\MyModule\Reports\OrdersReport;
use Aero\Admin\AdminReport;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        AdminReport::create(OrdersReport::class)
            ->title('Simple Orders Report')
            ->summary('View a simple report for your orders')
            ->permissions('orders.view');
    }
}
```

Your report class must implement a **newQuery** method that must return a **Illuminate\Database\Eloquent\Builder**. This method provides the query your report will run from.

```
<?php
```

```
namespace Acme\MyModule\Reports;

use Aero\Admin\Reports\Report;
use Aero\Cart\Models\Order;
use Illuminate\Database\Eloquent\Builder;

class OrdersReport extends Report
{
    protected function newQuery(): Builder
    {
        return Order::with([
            'status', 'items', 'currency', 'discounts', 'payments.method',
        ]);
    }
}
```

How do I add lenses to my custom report?

You need to create a class for your lens that extends **Aero\Admin\Reports\ReportLens** and implements the required public value function. The value function will return a string used to display your lens.

```
<?php

namespace Acme\MyModule\Reports\Lenses;

use Aero\Admin\Reports\ReportLens;

class TotalOrdersLens extends ReportLens
{
    protected function value()
    {
        return $this->query->count();
    }
}
```

You can optionally override the **render** method to change the view that your lens uses. The default **render** method:

```
public function render()
{
    return view('admin::reports.lens', ['title' => $this->title(), 'content' => $this->value()]);
}
```

You can also additionally override the **title** method to change the title shown on the frontend. By default this method returns a title that is generated from the lens class name.

Registering a Lens with your Report

To register your newly created lens with your report you need to ensure the report uses the **Aero\Admin\Reports\Traits\HasLens** trait and that your lens class is in the reports protected `$lenses` array.

```
<?php

namespace Acme\MyModule\Reports;

use Acme\MyModule\Reports\Lenses\TotalOrdersLens;
use Aero\Admin\Reports\Report;
use Aero\Admin\Reports\Traits\HasLens;
use Aero\Cart\Models\Order;
use Illuminate\Database\Eloquent\Builder;

class OrdersReport extends Report
{
    use HasLens;

    protected function newQuery(): Builder
    {
        return Order::with([
            'status', 'items', 'currency', 'discounts', 'payments.method',
        ]);
    }

    protected static $lenses = [
        TotalOrdersLens::class,
    ];
}
```

How do I add a table to my custom report?

You need to create a class for your table that extends **Aero\Admin\Reports\ReportTable** and implements the required public `columns` function. This `columns` function will be used to return an array containing your table's columns.

```
<?php

namespace Acme\MyModule\Reports\Tables;

use Aero\Admin\Reports\ReportTable;

class OrdersReportTable extends ReportTable
{
    public function columns(): array
    {
        return [

        ];
    }
}
```

Registering a Table with your Report

To register your newly created table with your report you need to ensure the report uses the **Aero\Admin\Reports\Traits\HasTable** trait and that your table class is in the report's protected static `$tables` array.

```
<?php

namespace Acme\MyModule\Reports;

use Acme\MyModule\Reports\Tables\OrdersReportTable;
```

```

use Aero\Admin\Reports\Report;
use Aero\Admin\Reports\Traits\HasTables;
use Aero\Cart\Models\Order;
use Illuminate\Database\Eloquent\Builder;

class OrdersReport extends Report
{
    use HasTables;

    protected function newQuery(): Builder
    {
        return Order::with([
            'status', 'items', 'currency', 'discounts', 'payments.method',
        ]);
    }

    protected static $tables = [
        OrdersReportTable::class,
    ];
}

```

Adding Columns to your Table

To add columns to your table you need to use the **Aero\Admin\Reports\ReportTableColumn** facade in your tables columns array method.

ReportTableColumn::create Parameters

Type	Description
String	The first parameter is the header of the column.
Closure	The second parameter is a closure that handles the display content for the column. The closure accepts a \$row variable and returns a string or view value.
String	Null

```

<?php

namespace Acme\MyModule\Reports\Tables;

```

```

use Aero\Admin\Reports\ReportTable;
use Aero\Admin\Reports\ReportTableColumn;

class OrdersReportTable extends ReportTable
{
    public function columns(): array
    {
        return [
            ReportTableColumn::create('Date', function ($row) {
                if ($row->ordered_at) {
                    return $row->ordered_at->format('D jS M, H:i');
                }

                return view('admin::reports.partials.muted-created-at', compact('row'));
            }),

            ReportTableColumn::create('Order', function ($row) {
                return $row->reference;
            }),
        ];
    }
}

```

Making your Column Searchable

You can use the **setSearchAction** method to define how your column is searched.

Type	Description
Closure	The first parameter is a closure that is called when your column is being used for a search. The closure is passed the query and search term.
String	Null

```

ReportTableColumn::create('Order', function ($row) {
    return $row->reference;
})
->setSearchAction(function ($query, $term) {
    $term = ltrim($term, '#');

```

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

Making your Column Not Exportable

To make your column not exportable you can use the **notExportable** method.

```
ReportTableColumn::create('Order', function ($row) {  
    return $row->reference;  
})  
->notExportable()
```

Setting your Columns Export Content

You can use the **setExportContent** method to define how your column should return data when exporting. This is useful for times where instead of returning a view on the Aero Admin you want your column to return raw data.

The **setExportContent** method accepts a closure that is passed the row.

```
ReportTableColumn::create('Date', function ($row) {  
    if ($row->ordered_at) {  
        return $row->ordered_at->format('D jS M, H:i');  
    }  
  
    return view('admin::reports.partials.muted-created-at', compact('row'));  
})  
->setExportContent(function ($row) {  
    if ($row->ordered_at) {  
        return $row->ordered_at->format('D jS M, H:i');  
    }  
  
    return $row->created_at->format('D jS M, H:i');  
})
```

Making your Column Invisible

You can use the invisible method to make your column invisible. Do note that your column will still be visible in an export.

```
ReportTableColumn::create('Order', function ($row) {  
    return $row->reference;  
})  
->invisible()
```

Setting your Columns Position

Columns are ordered by an Integer position value (lowest numbers go first). You can set a column's position using the position method. On the default Aero reports we increment each rows column.

```
ReportTableColumn::create('Order', function ($row) {  
    return $row->reference;  
})  
->position(1)
```

How do I add exporters to my custom report?

To add an exporter to your report you need to ensure the report uses the **Aero\Admin\Reports\Traits\CanExport** trait and that the exporter class is in the reports protected static \$exports array.

By default Aero provides a CSV, XML, and JSON exporter.

```
<?php

namespace Acme\MyModule\Reports;

use Aero\Admin\Reports\Exporters\CSVExporter;
use Aero\Admin\Reports\Exporters\JSONExporter;
use Aero\Admin\Reports\Exporters\XMLExporter;
use Aero\Admin\Reports\Report;
use Aero\Admin\Reports\Traits\CanExport;
use Aero\Cart\Models\Order;
use Illuminate\Database\Eloquent\Builder;

class OrdersReport extends Report
{
    use CanExport;

    protected function newQuery(): Builder
    {
        return Order::with([
            'status', 'items', 'currency', 'discounts', 'payments.method',
        ]);
    }

    protected static $exports = [
        CSVExporter::class,
        XMLExporter::class,
```

```
        JSONExporter::class,  
    ];  
}
```

Creating a Report Exporter

You need to create a class for your exporter that extends **Aero\Admin\Reports\ReportExporter** and implements the required public handle function.

Handle Parameters

Type	Description
Collection	The first parameter is a collection of the exportable tables columns.
Builder	The second parameter is the query that should be used to get the data for the export.
String	The third parameter is the key of the table being exported.

```
<?php  
  
namespace Acme\MyModule\Reports\Exporters;  
  
use Aero\Admin\Reports\ReportExporter;  
use Carbon\Carbon;  
  
class CustomExporter extends ReportExporter  
{  
    public function handle($columns, $rows, $key)  
    {  
        $file = $key.'-'.Carbon::now().'.csv';  
  
        return response()->streamDownload(function () use ($columns, $rows) {  
            $file = fopen('php://output', 'wb');  
  
            fputcsv($file, $columns->map->header()->toArray());  
  
            $rows->chunk(100, function ($chunk) use ($file, $columns) {  
                foreach ($chunk as $row) {
```

```
        $rowData = [];  
  
        foreach ($columns as $column) {  
            $rowData[] = $column->exportContent($row);  
        }  
  
        fputcsv($file, $rowData);  
    }  
});  
  
    fclose($file);  
}, $file,  
    ReportExporter::getHttpHeaders('text/csv', $file)  
);  
}  
}
```

Now you need to add your newly created exporter to your report as shown at the start of this article.

How do I extend existing reports?

Reports can be extended using a pipeline from your modules service providers setup method or your projects app service providers boot method. You can read more about pipelines [here \(link\)](#).

Adding a Column

To add columns to a table you can use the **addColumn**, **addColumnBefore**, or **addColumnAfter** methods from within the pipeline. These methods return a usual report table 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 when visiting the report.

```
<?php

namespace Acme\MyModule;

use Aero\Admin\Reports\Implementations\SalesReport;
```

```

use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        SalesReport::extend(function (SalesReport $report) {
            dd($report->table()->getColumns()->map->key());
        });
    }
}

```

```

<?php

namespace Acme\MyModule;

use Aero\Admin\Reports\Implementations\SalesReport;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        SalesReport::extend(function (SalesReport $report) {
            $report->table()->addColumn('My Column', function ($row) {
                return 'example';
            });
            ->setExportContent(function ($row) {
                return 'export content';
            });

            $report->table()->addColumnBefore('Before SKU', function ($row) {
                return $row->sku;
            }, 'sku');

            $report->table()->addColumnAfter('After Sku', function ($row) {
                return $row->sku;
            }, 'sku');
        });
    }
}

```

```
});  
}  
}
```

Adding a Lens

To add a lens to a report you can use the **addLens** method from within the pipeline. This method expects you to pass a report lens class. You can create a report lens class as shown in the article "

[How do I add Lenses to my custom Report?](#)

```
<?php  
  
namespace Acme\MyModule;  
  
use Acme\MyModule\Reports\Lenses\TotalRevenueLens;  
use Aero\Admin\Reports\Implementations\SalesReport;  
use Aero\Common\Providers\ModuleServiceProvider;  
  
class ServiceProvider extends ModuleServiceProvider  
{  
    public function setup()  
    {  
        SalesReport::extend(function (SalesReport $report) {  
            $report->addLens(TotalRevenueLens::class);  
        });  
    }  
}
```

Adding a Filter

To add a filter to a report 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 add Admin Filters to my custom Report?](#)

```
<?php
```

```
namespace Acme\MyModule;

use Aero\Admin\Filters\Order\OrderStatusAdminFilter;
use Aero\Admin\Reports\Implementations\OrderBreakdownReport;
use Aero\Common\Providers\ModuleServiceProvider;

class ServiceProvider extends ModuleServiceProvider
{
    public function setup()
    {
        OrderBreakdownReport::extend(function (OrderBreakdownReport $report) {
            $report->addFilter(OrderStatusAdminFilter::class);
        });
    }
}
```

How do I add admin filters to my custom report?

You need to create an Admin Filter class as shown in the article "[How do I create a custom report?](#)"

To register your newly created admin filter with your report you need to ensure the report uses the **Aero\Admin\Reports\Traits\HasFilter**trait and that your admin filter class is in the reports protected static \$filters array.

```
<?php

namespace Acme\MyModule\Reports;

use Aero\Admin\Filters\Order\OrderOrderedAtDateAdminFilter;
use Aero\Admin\Reports\Report;
use Aero\Admin\Reports\Traits\HasFilters;
use Aero\Cart\Models\Order;
use Illuminate\Database\Eloquent\Builder;

class OrdersReport extends Report
{
    use HasFilters;

    protected function newQuery(): Builder
    {
        return Order::with([
            'status', 'items', 'currency', 'discounts', 'payments.method',
        ]);
    }

    protected static $filters = [
        OrderOrderedAtDateAdminFilter::class,
    ];
}
```