# Paginate results

**Tutorial path**

By default, the Search Service returns 25 results so, without pagination, the current interface allows you to view only the 25 Locations. To add the pagination, we need to modify the whole component *stack* to accept an offset parameter and to use it:

- the Symfony Controller and the routing configuration should accept an `offset` parameter and the search query should be built with it
- the JavaScript application should also have a route with an `offset` parameter
- this `offset` should be used in the view service when doing the AJAX request
- the server side code should generate the pagination links and the view should handle those links as the Location links in the previous step.

## Symfony controller and routing configuration to accept an `offset` parameter

As you can see in the corresponding commit, this is a purely Symfony-related task where we have to modify the `routing.yml` to accept an optional parameter and the action to use it.

> This is detailed in the Symfony Controller documentation.

## JavaScript application routing to accept an `offset` parameter

The PlatformUI application does not support route with optional parameters, so as a result we are forced to declare a new route which will accept the new parameter. To do that, we have to modify the application plugin to add a second route called `eZConfListOffset` :

**ezconf-listappplugin.js**

```
// adding a new route so that we don't have anything
else to change
            // and we can manage the default `offset`
value in the view service
            app.route({
                name: "eZConfListOffset",
                path: "/ezconf/list/:offset",
                view: "ezconfListView",
                service: Y.eZConf.ListViewService,
                sideViews: {'navigationHub': true,
'discoveryBar': false},
                callbacks: ['open', 'checkUser',
'handleSideViews', 'handleMainView'],
            });
```

The new route is very similar to the existing one with the exception of its `path` property.

> The route placeholder concept is documented in the YUI Router component.

## Change the view service to use the `offset` parameter

Depending on which route is matched, an offset parameter might be available. The matched route parameters are available in the request object stored in the `request` attribute of the view service. So to work with both the `eZConfList` route and the `eZConfListOffset` route, the view service has to check if an offset was passed and to use 0 as its default value if none is provided. Once that is done, it can build the URI to use to do the AJAX request. The `_load` method then becomes:

**_load method in ezconf-listviewservice.js**

```
_load: function (callback) {
            // the request allows to retrieve the
matched parameters
            var offset =
this.get('request').params.offset,
                uri;

            if ( !offset ) {
                offset = 0;
            }
            uri = this.get('app').get('apiRoot') +
'list/' + offset;

            Y.io(uri, {
                method: 'GET',
                on: {
                    success: function (tId, response) {
                        this._parseResponse(response);
                        callback(this);
                    },
                    failure: this._handleLoadFailure,
                },
                context: this,
            });
        },
```

At this point, the interface in the browser should remain the same, but by using for instance the URL `/ez#/ezconf/list/10`, you check that the offset is correctly taken into account.

## Pagination links

To have working pagination links, the first thing to do is to change the server side code to generate them. In the corresponding commit we also define the default limit at 10 elements. Like for the Location links, the server side code is not really able to generate a link in the JavaScript application, so we have to generate the markup with the offset as metadata so that the view can recognize and correctly handle those links. So those links get a `data-offset` attribute with the corresponding offset:

```
{% block content %}
<!-- [...] this list is generated, removed here to keep
this code short -->

<ul class="ezconf-list-pagination">
    <li class="ezconf-list-page ezconf-list-previous">
    {% if previous is not same as(false) %}
        <a href="{{ path('list', {offset: previous}) }}"
class="ezconf-list-page-link" data-offset="{{ previous
}}">&laquo; Previous</a>
    {% else %}
        <span>&laquo; Previous</span>
    {% endif %}
    </li>
    <li class="ezconf-list-page ezconf-list-next">
    {% if next %}
        <a href="{{ path('list', {offset: next}) }}"
class="ezconf-list-page-link" data-offset="{{ next
}}">Next &raquo;</a>
    {% else %}
        <span>Next &raquo;</span>
    {% endif %}
    </li>
</ul>
{% endblock %}
```

After, we just have to change the view code to handle a *tap* on the next/previous links and when this happens, we can again fire the `navigateTo` application level event to ask the view service to trigger the navigation but this time to the `eZConfListOffset` route with the given `offset`, this is done with:

**Pagination links handling in ezconf-listview.js**

```
YUI.add('ezconf-listview', function (Y) {
    Y.namespace('eZConf');

    Y.eZConf.ListView = Y.Base.create('ezconfListView',
Y.eZ.ServerSideView, [], {
        events: {
            '.ezconf-list-location': {
                'tap': '_navigateToLocation'
            },
            '.ezconf-list-page-link': {
                'tap': '_navigateToOffset' // new event
detected
            },
        },

        _navigateToOffset: function (e) {
            var offset = e.target.getData('offset');

            e.preventDefault();
            this.fire('navigateTo', {
                routeName: 'eZConfListOffset',
                routeParams: {
                    offset: offset,
                },
            });
        },

        // ... the rest remains unchanged, removed to
keep the code short
    });
});
```

After this change, the pagination should work as expected and you should be able to navigate through the complete list of Locations.

**Results and next step**
The resulting code can be seen in the `7_pagination` tag on GitHub, this step result can also be viewed as a diff between tags `6_2_list_server` and `7_pagination`.

The next and final step is to add a way for the user to filter by Content Type.