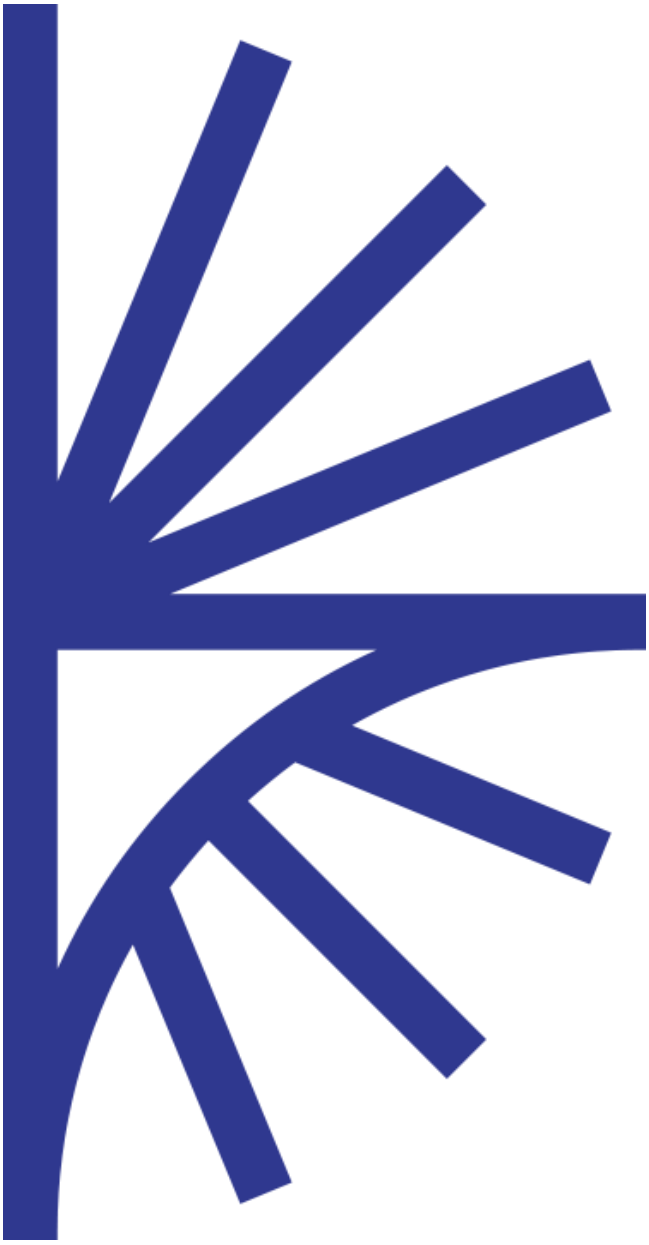


FUSION REGISTRY

User Interface Extensions

FUSION REGISTRY
VERSION 9

Tutorial

This guide describes how to create custom pages in the Fusion Registry Web Application

Fusion Registry: 9.4.x
Document Last Updated: 15th Jan 2019

Version: 20190115

Contents

1 Overview

This document discusses how to create custom pages in the Fusion Registry web User Interface. An extension is simply a new HTML page which is accessed via the Fusion Registry web application. An extension can be made up of one or more HTML files. Extensions can include JavaScript libraries. Metadata Technologies' IMJS framework is automatically added and initialised for each Extension, making it possibly to easily communicate with the Fusion Registry SDMX API. Extensions can be protected by making use of the Fusion Registry security framework to restrict access to certain users.

2 Creating an Extension

In the Fusion Registry home folder[1] create a Sub Folder of Extensions.

<User Home>\MetadataTechnology\FusionRegistry\Extensions

The Extensions folder is the root folder for all extensions, to create an extension create a sub-folder to hold the extension HTML and Javascript, in this example an extension called Reports will be created.

<User Home>\MetadataTechnology\FusionRegistry\Extensions\Reports

The <User Home> directory is obtained from the Java Environment Variable **user.home**. This defaults to the home location for the user who started the Fusion Registry. The user.home environment variable can be changed using a Java VM Argument.

In Apache Tomcat VM Arguments can be passed by creating a setenv.bat (windows) or setenv.sh (unix) file under the bin folder. The following can be put into this file.

```
set "JAVA_OPTS=%JAVA_OPTS% -Duser.home=<new_location>"
```

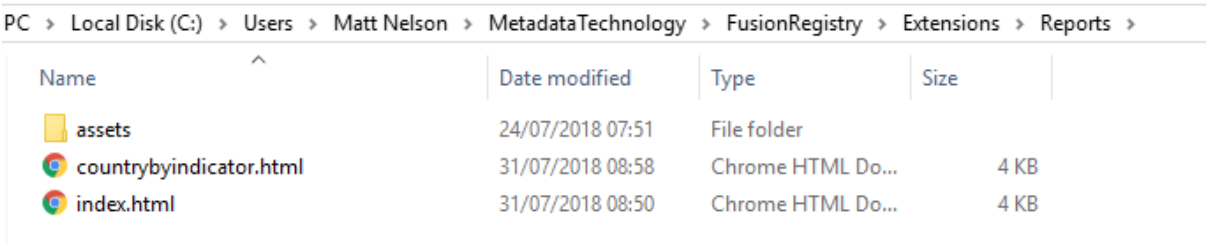


Figure 1 showing an extension which generates reports from the Fusion Registry content

In the Reports extension example, JavaScript libraries have been included for common frameworks such as bootstrap, JQuery, and font-awesome, this is shown in the image below.

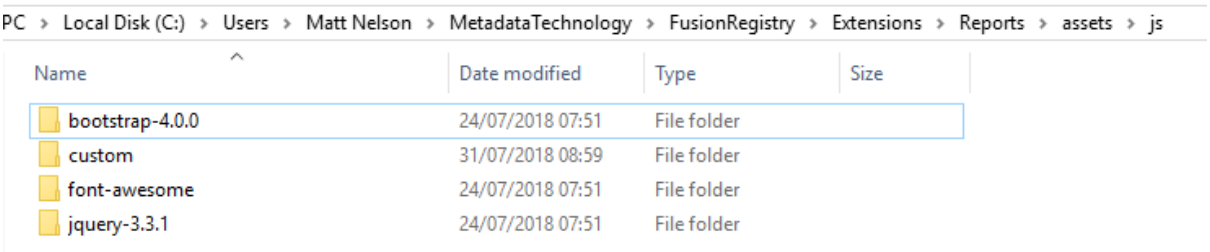


Figure 2 showing the JavaScript libraries included in the Reports extension

The custom HTML page can be built using standard HML constructs. In addition there are two additional variables which can be used:

Variable Name	Purpose
${baseUrl}	This is replaced at runtime to be the URL of the Registry server. The URL value is set on Registry installation, and can be modified under the Registry Server Settings. It is possible to postfix

[1] See the Fusion Registry Set Up Guide for details on the Home Folder

	this variable to create absolute paths, for example ${baseURL}/img/someImage.png
${imjs}	This variable includes and initialises the IMJS library. Any custom JavaScript can use the library by using the global variable **imjs, o**r by using the static accessor **IMJS.getInstance()**. It is important to include this variable along with the other scripts, as this will be replaced by <Script> tags

Figure 3 showing the Reports HTML page, which includes a link to the Registry home page, and include the IMJS framework

On Tomcat start-up any folders under the Extensions folder will be copied into the Fusion Registry web application, and made available in the UI under the following path:

http(s)://[Fusion Registry URL]/ex/[Folder Name]

In the previous example, testing locally this will be:

http://localhost:8080/FusionRegistry/ex/Reports/index.html

It is possible, in development to modify the files in this location and copy them back to the Fusion Registry Home folder. When the Tomcat instance is re-started, the folder will be re-copied across, overwriting any changes made directly in the web application.

3 Replacing the Fusion Registry Landing Page

To create an extension which replaces the default Fusion Registry landing page, the extension folder must be called ROOT.

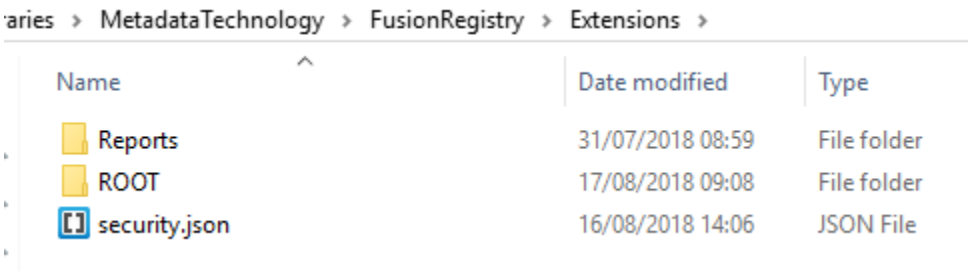
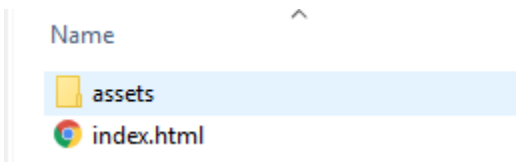


Figure 4 showing the ROOT folder

The ROOT folder must contain an index.html file, which will become the new landing page of the Fusion Registry.

Unlike other Extension folders, where included libraries can be relative links, files under the ROOT folder must link to subfolders using the prefix 'ex/ROOT'. For example the following ROOT folder



Must use the relative link 'ex/ROOT/assets' to reference assets from the index.html file, as shown below.

```
<script src="ex/ROOT/assets/js/jquery-3.3.1/jquery-3.3.1.min.js"></script>
<script src="ex/ROOT/assets/js/bootstrap-4.0.0/bootstrap.min.js"></script>
<script src="ex/ROOT/assets/js/imjs/imjs.min.js"></script>
<script src="ex/ROOT/assets/js/font-awesome/js/fontawesome-all.min.js"></script>
<script src="ex/ROOT/assets/js/dropzone/dropzone.js"></script>
<script src="ex/ROOT/assets/js/custom/index.js"></script>
```

Figure 5 showing relative links from index.html in the ROOT folder, using the ex/ROOT path prefix

The reason for this modification is that the ROOT folder (and all other Extensions) will be copied by the Fusion Registry on tomcat startup to a folder named **ex** in the tomcat/webapps/FusionRegistry web application, and the Registry needs to know to route these relative URLs to the ROOT extension.

4 Extensions and IMJS

As noted in the previous section, the IMJS Framework can be used to easily access the SDMX Metadata and Data in the Fusion Registry. If using the ${imjs} variable in the HTML page, the Fusion Registry will include the IMJS JavaScript library and configure it to use the public SDMX API.

IMJS documentation can be found online from the following URL:

https://metadatatechnology.com/standalone/IMJS/index.html

The following image shows IMJS being used to obtain all Dataflows that have data associated with them, and obtain all Data Structure Definitions.

```
$(document).ready(function () {
    var h = $("body").height() - 300;
    $("#table-container").height(h);

    imjs.getDataflows(null, null, null, "only", function (dfs) {
        imjs.getDataStructures(null, null, null, false, function (dsds) {
            var toQuery = {};
            dsds.forEach(function (dsd) {
                if (dsd.getDimension("INDICATOR")) { ··· }
            });
            for (var urn in toQuery) { ··· }

            $("#search").removeClass("disabled");
            $("#search").click(performSearch);
        });
    });
});
```

Figure 6 showing IMJS being used to obtain Dataflows and Data Structures before processing the response

The imjs variable refers to the following class which can be used to obtain all structural metadata from the Fusion Registry:

https://metadatatechnology.com/standalone/IMJS/api.IMJS.html

5 Security

It is possible to secure Extensions to only allow access to users with given roles. To secure an extension, create a file called **security.json** under the Extensions folder.

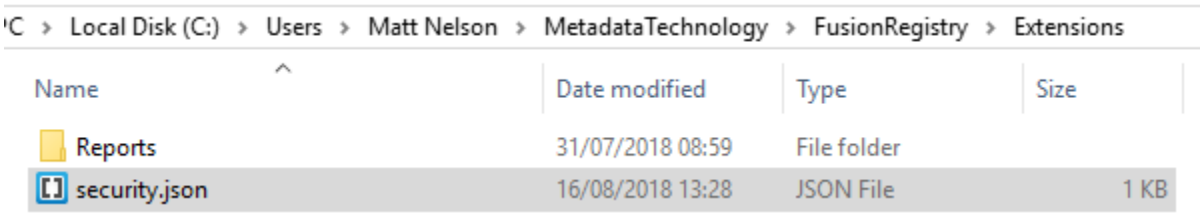


Figure 7 showing the security.json file, defining the security rules

The security.json file defines the user roles that are allowed for each Extension. In this case, the Extension sub-folder 'Reports' is only accessible from a user with the role ROLE_ADMIN.

```
{
    "Reports": {
        "Roles": ["ROLE_ADMIN"]
    }
}
```

Figure 8 showing an example security.json file contents

Other roles include:

ROLE_ADMIN
ROLE_AGENCY
ROLE_DATACONSUMER
ROLE_ DATAPROVIDER

An Extension can support multiple roles, for example

```
{
    "Reports": {
        "Roles": ["ROLE_ADMIN","ROLE_AGENCY"]
    }
}
```

Figure 9 showing multiple roles supported for the Reports Extension

If security.json file is not present, or the Extensions folder is not included in the security.json file, then there will be no security applied to the Extension.

If a user attempts to access an Extension without having the correct role, then they will be presented with a HTTP 404 (Not Found) page.