Mediaflux Portals

This document describes how Portal pages are composed.

Status

A summary of the status of sections of this document, in the opinions of the authors. When authors agree, the status becomes true.

Section	**Author**	**Status**					
		Draft / high churn					
			Draft / refinements				
				Ready for approval			
					Approved		
						Implemented	
							Tested
Estimates	Daniel				▼		
	Lewis					▲	
Configuration	Daniel				▼		
	Lewis					▲	
Components: Page layout	Daniel				▼		
	Lewis					▲	
Components: List page	Daniel				▼		
	Lewis					▲	
Components: Forms and Dialogs	Daniel				▼		
	Lewis					▲	
Behaviour	Daniel				▼		
	Lewis				▲		

Terms

This section is included to make the remainder of the document more concise.

End User (or just **User**): a person who looks at a portal with their web browser, ideally not aware of MediaFlux etc

Portal Designer / Portal Author: a person who is "making" a portal -- authoring a portal config xml file, authoring CSS to control the portal's presentation, and/or authoring JavaScript to enhance portal behaviour

`config.portal.<foo>.<bar>`: refers to content in the XML document describing the portal. The XML document has a `<portal>` element as the root node, and the following items in the dot-separated list indicate descending into child nodes

Facts

This section captures facts that have been identified as valuable information.

A page always has the header and navbar, then content, then footer. The content is one of three content types:

- Static content
- List of collections
- List of assets

Questions/proposals

This section is for capturing moment-to-moment queries between document authors. It is certain to change as answers are added to the Facts section or are integrated into the document in other ways.

Curiosity Q: Do you reckon we can rule that config XML should always be in UTF8? As in, require the first line to be

```
<?xml version="1.0" encoding="UTF-8"?>
```

Instead of

```
<?xml version="1.0"?>
```

The cat example file is full of hexadecimal entity references representing greek letters etc, like κάττα written as κάττα … is it okay to lock in utf8 you reckon?

Q: if we have a sub-namespace for a portal's JS and CSS, can I also add supporting items like the logo and background images, and get a path to those to include in my CSS and header/footer content?

A: a resource folder sure. In Mediaflux you can make a file public that can be referenced in your CSS. It sure is feasible just not sure how exactly it should work. Let's make it optional at this stage.

Estimates

This section is Lewis' current estimates for performing the work described in this document.

Restructuring Portal in flat HTML, styling it in CSS, adding JS trigger event

Five days including scoping.

Potential portal paging re-design

One day (approx.)

Lewis proposes that we deliver the styling portal in CSS task first then we may circle back to improve the paging.

Configuration

This section describes how portals are configured in MediaFlux, primarily via the portal definition XML file.

Config section: preamble

The start of the portal config includes the xml declaration, opens the root `portal` tag, and gives some portal level metadata.

```
<?xml version="1.0" encoding="UTF-8"?>
<portal>
   <title>Cat Breeds</title>
```

Auth information. Check Architecta docs for correct configuration here.

```
   <authentication>
      <domain>system</domain>
      <user>manager</user>
      <password>change_me</password>
      <!--
      <domain>portal</domain>
      <user>anonymous</user>
      <password>password1</password>
      -->
      <!--
<token>DbVXGhATACT7jQnaLwBPfetRleoXLFlbNfLM6Jvo2cdSLxtGN9w8EDhbRATgPQLcOARNRX75zls
JsNaY4e21tYXnxK81Yq3Q9EEtbzEURPeRDvNRkIqaSQ6Bj494V1014361</token>
      -->
   </authentication>
```

What to say (if anything) when the user attempts to leave the page

```
   <leave>
      <check>false</check>
      <message>Please don't leave!</message>
   </leave>
```

The root query that selects the assets available to this portal

```
   <query>namespace>='/arcitecta/test-data/animals'</query>
```

Optionally, a set of styles. These have been superseded by portal specific CSS.

```
<style name="collection-result">
   ...
</style>
<style name="result">
   ...
</style>
```

Configuration: portal owner CSS and JS

The `<configuration>` tag contains `<css>` and `<javascript>` tags, which specify CSS and Javascript content that will be included in the portal. Each tag contains the path to the asset starting from `/www/aportal/portals`.

```
<configuration>
   <css>basic.css</css>
   <css>custom.css</css>
   <css>mediaplayer.css</css>
   <javascript>fixlayout.js</javascript>
   <javascript>mediaplayer.js</javascript>
</configuration>
```

The assets will be referenced from the portal HTML using `<link>` and `<script>` tags as appropriate, in the order they appear in the `<configuration>`.

If the `<configuration>` tag does not include at least one `<css>` tag, the portal will attempt to include CSS rules from the `/www/aportal/portals/mf-portal-default-css` asset. There is no default javascript (however note that portals are assembled using the GWT toolkit which uses javascript).

Config section: forms

Presently the only form used by portals is the "create new asset" form.

The `xpath` attributes on each `xelement` indicate the source of each creation field in the form.

```
<form name="breed">
   <xelement xpath="arc.test:cat">
      <vertical>
         <horizontal>
            <label>Breed</label>
            <field><xelement xpath="breed" /></field>
         </horizontal>
         <horizontal>
            <label>Origin</label>
            <field><xelement xpath="origin" /></field>
         </horizontal>
         <horizontal>
            <label>Source</label>
            <field><xelement xpath="source" /></field>
         </horizontal>
         <horizontal>
            <label>Body Shape</label>
            <field><xelement xpath="body" /></field>
         </horizontal>
         <horizontal>
            <label>Coat</label>
            <field><xelement xpath="coat" /></field>
         </horizontal>
         <horizontal>
            <label>Pattern</label>
            <field><xelement xpath="pattern" /></field>
         </horizontal>
      </vertical>
   </xelement>
</form>
```

Config section: layout

This section describes aspects of the portal's layout and rendering. This section is contained within a single `layout` tag.

```
<layout>
```

The header allows specification of the portal header, present on every page. The contents of `portal.layout.header.content` are copied into the portal's `<header>` HTML tag (see *Component: Header* later in this document). Note that this means your header HTML must also be valid XML.

The img and h1 tags in this example include CSS classes that are addressed by the default portal CSS. Using those specific classes will mean you benefit from that work.

```
<header>
   <content>
      <img class="mf-header-logo" src="path/cats-logo.png">
      <h1 class="mf-heading">
         Data Access Portal
      </h1>
   </content>
</header>
```

The menu tag includes static page content for the portal, such as introductory or help text, FAQs, contact details etc. Like the header, everything in the content tag gets copied into the portal page. HTML is okay but remember that your HTML needs to be valid XML.

Note the `default="true"` attribute, which marks a static page as the content a user sees immediately when visiting the portal.

```
<menu>
   <entry label="FAQ">
      <title>Frequently Asked Questions</title>
      <content>
         <p>The domestic cat ...</p>
         <p>In comparison to ...</p>
      </content>
   </entry>
   <entry label="About" default="true">
      <title>Welcome to the Cat Breeds Database</title>
      <content>
         <p>...</p>
      </content>
   </entry>
</menu>
```

The footer is similar to the header, in that it simply supplies content to fill the `<footer>` tag in the portal's web page.

```
<footer>
   <content>
       Powered by Mediaflux&#174;
   </content>
</footer>
```

Finally, the closing layout tag:

```
</layout>
```

Config section: collections

A sequence of collection tags describing the collections available through this portal.

Refer to Architecta documentation for how to describe collections correctly. Certain sections are included here for comment.

Note that the `of="collections"` attribute in this example describes this as a collection containing collections.

```
<collection label="Collections" of="collections">
   <title>Cats by Coat</title>
   <restricted>
      ...
   </restricted>
   <search>
      <!-- <query>type='arc/test/animal/collection'</query> -->
      <facets>
         ...
      </facets>
      <cursor-size>10</cursor-size>
      <layout>
         <results>
            <result>
               <icon></icon>
               <title>
                  <xpath>name</xpath>
               </title>
               <description>
                  <xpath>description</xpath>
               </description>

               <!-- You can retrieve other details that are not displayed. -->
               <!-- Those details can be used in the description when -->
               <!-- displaying the members of a collection -->
               <detail visible="false">
                  <xpath>meta/mf-note/note</xpath>
               </detail>
            </result>
         </results>
      </layout>
   </search>

   <!-- Alternate display for the collection members -->
   <member min-level="1" max-level="1">

      <description>
         <!-- The following xpath must have been declared -->
         <!-- in the search/layout/results section above. -->
         <!-- This may include xpaths that were denoted -->
         <!-- as non-visible. -->
         <xpath>description</xpath>
      </description>
```

The facets listed here are used as refining options to narrow the selected set of collection members.

```
<search>
   <facets>
      <text>
         <description>Search for cats...</description>
      </text>
      <filters label="Format">
         <mode>single</mode>
         <description>The type of cat</description>
         <filter label="Short Haired">
            <query>(xpath(arc.test:cat/coat) = 'Short')</query>
         </filter>
         ...
         <filter label="Hairless">
            <query>(xpath(arc.test:cat/coat) = 'Hairless')</query>
         </filter>
      </filters>
   </facets>
```

How many collection members per page.

```
<cursor-size>20</cursor-size>
```

This layout defines the set of fields that will be included in the portal.

```
      <layout>
        <results>
          <result>
            <icon></icon>
            <title>
              <xpath>name</xpath>
            </title>
            <detail label="Origin">
              <xpath>meta/arc.test:cat/origin</xpath>
            </detail>
            <detail label="Source">
              ...
            </detail>
            <detail label="Pattern">
              <xpath>meta/arc.test:cat/pattern</xpath>
            </detail>
          </result>
          <action label="Do Something"></action>
        </results>
      </layout>
    </search>

    <egest>
      <restricted>
        <role>user</role>
      </restricted>
    </egest>

    <create>
      <restricted>
        <role>user</role>
      </restricted>

      <namespace>/arcitecta/test-data/animals</namespace>

      <name/>
      <description/>
      <geometry/>

      <metadata>
        <form-name>breed</form-name>
      </metadata>

      <content>
        <extension>jpg</extension>
        <extension>png</extension>
        <max-number>1</max-number>
        <max-size>40000</max-size>
      </content>

    </create>

  </member>
  <create>...</create>
  <egest>...</egest>
</collection>
```

Here is another collection, this time containing assets.

```
<collection label="Breeds" of="assets">
   <title>Cat Breeds</title>
   <search>
      <query>(xpath(arc.test:cat) has value)</query>
      <sort order="asc">meta/arc.test:cat/breed</sort>

      <facets>
         ...
      </facets>
      <cursor-size>20</cursor-size>
      <layout>
         <results>
            <result>
               <icon></icon>
               <title>
                  <xpath>meta/arc.test:cat/breed</xpath>
               </title>
               <detail label="Origin">
                  <xpath>meta/arc.test:cat/origin</xpath>
               </detail>
               <detail label="Source">
                  <xpath>meta/arc.test:cat/source</xpath>
               </detail>
               <detail label="Body Shape">
                  <xpath>meta/arc.test:cat/body</xpath>
               </detail>
               <detail label="Coat">
                  <xpath>meta/arc.test:cat/coat</xpath>
               </detail>
```

Note this next detail includes a `visible="false"` attribute, meaning it will be included in the data island for this asset, but not shown to the user (refer to the *Component: AssetInfoBlock* section for more information).

```
            <detail label="Pattern" visible="false">
               <xpath>meta/arc.test:cat/pattern</xpath>
            </detail>
         </result>
         <action label="Do Something"></action>
      </results>
   </layout>
</search>
<create>
   <restricted>
      <role>user</role>
   </restricted>
   <namespace>/arcitecta/test-data/animals</namespace>

   <name/>
   <description/>
   <geometry/>

   <metadata>
      <form-name>breed</form-name>
   </metadata>

   <content>
      <extension>jpg</extension>
      <extension>png</extension>
      <max-number>1</max-number>
      <max-size>40000</max-size>
   </content>

</create>
<egest>
   <restricted>
      <role>user</role>
   </restricted>
</egest>
</collection>
```

Finally, the closing portal tag ends the config file.

```
</portal>
```

Page meta-information

The <head> tag delivered by the portals interface should include this meta tag:

```
<meta name="viewport" content="width=device-width, minimum-scale=1,
initial-scale=1">
```

Page layout components

This section lists HTML components used to construct the broad layout / structure of portal pages. Source examples will include literal HTML, and component names nested in {{ double curly brackets }} indicating that the component's source will be inserted there.

Root Component: PageStructure

The root layout used to present all page types.

Status: proposal

Components included

- Header
- NavigationBar
- *one of:*
    - StaticPage
    - ListPage/Collections
    - ListPage/Assets
- Footer

Example source

```
<body>
<div class="mf-body-wrapper">
   {{ Header }}
   {{ NavigationBar }}
   {{ StaticPage |or| ListPage/Collections |or| ListPage/Assets }}
   {{ Footer }}
</div>
</body>
```

Component: Header

Portal header. The content within the `<header>` tag is copied directly from `config.portal.layout.header.content` – the portal designer can put whatever they want there (as long as it's well formed XML).

The default content includes an image and `h1` heading that include the classes shown in the example below, and the default portal CSS includes styling to support those classes, but a portal designer is not required to use that content or classes.

TODO devise generic title / logo image for default/sample portal XML

TODO add generic logo image to default portal configuration file

TODO add content to default portal configuration file

TODO update example source below to match sample

Status: proposal

Example source

```
<header class="mf-header" role="banner">
   <img class="mf-header-logo" src="path/cats-logo.png">
   <div class="mf-heading" role="heading">
      Data Access Portal
   </div>
</header>
```

Component: Footer

Portal footer. The content within the `<footer>` tag is copied directly from `config.portal.layout.footer.content` – the portal designer can put whatever they want there (as long as it's well formed XML).

The default content includes (*TODO: things Daniel will write*) that include the classes shown in the example below, and the default portal CSS includes styling to support those classes, but a portal designer is not required to use that content or classes.

TODO devise generic footer content for default/sample portal XML, ideally including a CC-BY licence or something like that

TODO add content to default portal configuration file

TODO update example source below to match sample

Status: proposal

Example source

```
<footer class="mf-footer">
   {{ config: portal.layout.footer.content }}
</footer>
```

Component: NavigationBar

Navigation menu, offering static and collection pages.

The items listed are each collection (each `config.portal.Collection`, named by the `label` attribute of the `Collection` tag), followed by each static page (each `config.layout.menu.entry`, named by the `label` attribute of the `entry` tag).

Clicking any item other than the current one results in the page's content being replaced with the selected content.

Status: proposal

Components included: none

Example source

```
<nav class="mf-navbar">
   <div class="mf-navbar-container">
      <ul class="mf-nav-list">
         <li><a href="">Collections</a></li>
         <li class="mf-current-nav-item"><a href="">Breeds</a></li>
         <li><a href="">FAQ</a></li>
         <li><a href="">About</a></li>
      </ul>
   </div>
</nav>
```

Component: StaticPage

A static page. The title and content is defined by the portal definition

Status: drafted by Lewis

Example source

```
<div class="mf-page mf-static-page" role="main" aria-labelledby="mf-page-title">
   <h1 class="mf-static-page-title" id="mf-page-title">Frequently Asked
Questions</h1>
   <div class="mf-static-page-content">
      …
   </div>
</div>
```

List page components

These components are primarily used to list and display collections and assets and related control panels

Component: ListPage

A page that is used to display a list of something. There are two variants; ListPage/Collections is used to display a list of asset collections, and ListPage/Assets is used to display a list of assets themselves.

Components included

- CollectionBar
- CollectionSideBar
- PagingControl
- AssetCreationForm (a part of a ListPage, but actually present as an immediate child of the html body tag)
- *one of:*
    - CollectionList
    - AssetList

Example source

```
<div class="mf-page mf-list-page" aria-labelledby="mf-page-title" role="main">
   {{ CollectionBar }}
    <div class="mf-list-page-content">
         {{ CollectionSideBar }}
         {{ PagingControl }}
         {{ CollectionList |or| AssetList }}
   </div>
</div>
{{ AssetCreationForm (as a child of the body tag) }}
```

Component: CollectionBar

Collection page top bar, including page title and a couple of page actions: search box and asset creation button.

Status: drafted by Lewis

Example source

```
<div class="mf-collection-bar">
   <h1 class="mf-collection-title" id="mf-page-title">Short Haired Cats</h1>
   <div class="mf-collection-actions">
      <form class="mf-action mf-action-search" role="search">
         <input type="search" placeholder="search for cats...">
      </form>
      <div class="mf-action mf-action-button">
         <button class="mf-button mf-new-item-button">
            new
         </button>
      </div>
   </div>
</div>
```

Component: CollectionSideBar

Collection page side-bar, including a group of facet selectors and a title.

The filter mode can be “single” or “multi”, which applies radio or multiple selection style respectively.

Status: Lewis

Example source

```
<div class="mf-collection-sidebar">
   <div class="mf-facet-groups">
        <form role="search">
           <fieldset class="mf-facet-selector">
              <legend class="mf-facet-title">Format</legend>
              <label class="mf-facet mf-checkbox/mf-radio">
                 <input type="checkbox/radio" name="format">
                 <span>Short Haired</span>
                 </label>
              <label class="mf-facet mf-checkbox/mf-radio">
                 <input type="checkbox/radio" name="format">
                 <span>Medium Hair</span>
              </label>
              ...
           </fieldset>
        </form>
   </div>
</div>
```

Alternative proposal from Daniel - fieldsets for the various facets

```
<div class="mf-collection-sidebar">
   <form role="search">

      <fieldset class="mf-facet-selector">
         <legend class="mf-facet-title">Hair length</legend>
         <label class="mf-facet mf-checkbox/mf-radio">
            <input type="checkbox/radio" name="format">
            <span>Short Haired</span>
         </label>
         <label class="mf-facet mf-checkbox/mf-radio">
            <input type="checkbox/radio" name="format">
            <span>Medium Hair</span>
         </label>
         ...
      </fieldset>

      <fieldset class="mf-facet-selector">
         <legend class="mf-facet-title">Another facet</legend>
         <label class="mf-facet mf-checkbox/mf-radio">
            <input type="checkbox/radio" name="...">
            <span>facet value 1</span>
         </label>
         <label class="mf-facet mf-checkbox/mf-radio">
            <input type="checkbox/radio" name="...">
            <span>facet option 2</span>
         </label>
         ...
      </fieldset>

   </form>
</div>
```

Component: PagingControl

Collection paging control. The behaviour of this control is a little odd. The control will show buttons in this order:

- a goto-first-page button (refresh icon)
- a previous-page button (not shown if the current page is the first page)
- a goto-page-X-button for each page, from page 1 up to the current page
- a button for the current page (not clickable)
- a next-page button (not shown if the current page is the last page)

This means that on page 2, there will be two buttons with page numbers: [1] [2]. On page 23, there will be 23 buttons with page numbers: [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22] [23].

(Note: it's possible – but deeply wrong – to hide some of the too-many-buttons with CSS.)

Optional enhancement

The user expectation is probably more like:

[<< first] [< prev] [5] [6] **7** [8] [9] [next >] [last >>]

..with a group of five clickable page numbers centered on the current page. Maybe there's no first / last links, if they're hard to calculate, e.g.

[<-- previous] [1] … [5] [6] **7** [8] [9] … [99] [next -->]

Status: proposal

Example source

```
<nav class="mf-collection-paging" aria-label="paging">
   <ul>
      <li>
         <a class="mf-button mf-up-button">up</a>
      </li>
      <li>
         <a class="mf-button mf-refresh-page-button">refresh</a>
      </li>
      <li>
         <a class="mf-button mf-next-prev-button">prev</a>
      </li>
      <li>
         <a class="mf-button mf-page-button mf-current-page-button">1</a>
      </li>
      <li>
         <a class="mf-button mf-page-button">2</a>
      </li>
      <li>
         <a class="mf-button mf-next-page-button">next</a>
      </li>
   </ul>
</nav>
```

Component: ModalDialog

Used to display content, for example a form, popped-up over the top of the normal content. At present the only content presented this way is the AssetCreationForm.

An additional requirement of displaying a modal dialog is to add a class to the page's body tag when a modal dialog is displayed, and remove that class when the model dialog is hidden.

An example of this form construction is available at:

http://danielbaird.com/mf-portal-modern/form/newform.html

Components included

- AssetCreationForm
- Video/AudioPlayer

Status: drafted by Daniel

Example source

```
<div class="mf-modal-wrapper">
   <div class="mf-modal-container" role="dialog">
      {{ AssetCreationForm |or| Video/AudioPlayer }}
   </div>
</div>
```

Component: Video/AudioPlayer

Example source for Video

```
<video class="mf-media-player" controls>
  <source src="movie.mp4r" type="video/mp4">
  Your browser does not support the video tag.
</video>
```

Example source for Audio

```
<audio class="mf-media-player" controls>
  <source src="horse.ogg" type="audio/ogg">
  Your browser does not support the audio tag.
</audio>
```

Component: AssetCreationForm

Popup form window to create a new asset. There are three sections of the form, each wrapped in a `<fieldset class="mf-form-fieldset">` tag: first, a section for collecting the asset metadata including name; a section collecting the asset file; and finally a section containing the cancel and create action buttons.

This component could be divided into more sub-components (e.g. a component to represent the sectioning fieldsets, a component to represent the dialog management buttons) but for brevity, only sub-components that are provisionally included are split out here.

Note that the file upload section should not include a FileUploadTable until a file is added by the user.

Components included

- GeneratedMetadataCollectionForm
- FileUploadTable

Status: drafted by Daniel

Example source

```
<form class="mf-form">
   <fieldset class="mf-form-fieldset mf-asset-metadata-fieldset"
aria-label="metadata">
      {{ GeneratedMetadataCollectionForm }}
   </fieldset>

   <fieldset class="mf-form-item mf-file-upload-item">
       <div class="mf-file-upload-info">
           <label class="mf-form-item-label-wrap">
             <span class="mf-form-item-prompt">Files to upload</span>
           </label>
           <div class="mf-field mf-file-upload">
             {{ zero or one FileUploadTable }}
           </div>
       </div>
     <button class="mf-button mf-clear-files-button">clear files</button>
   </fieldset>

   <fieldset class="mf-form-item mf-form-actions-item" aria-label="actions">
     <button class="mf-button mf-cancel-button">cancel</button>
     <button class="mf-button mf-create-button">create</button>
   </fieldset>
</form>
```

Aspirational alternative source for the first fieldset

This alternative to the metadata fieldset is not part of the implementation plan due to constraints. It is preserved here for potential use for future projects - see Appendix for details.

```
   <fieldset class="mf-form-fieldset mf-asset-metadata-fieldset">
     {{ FormTextItem/TitleItem }}
     {{ zero of more FormSelectionItem and/or FormTextItem  }}
   </fieldset>
```

Component: FileUploadTable

This is used inside the file upload portion of an AssetCreationForm, when a file has been selected by the user. In principle, multiple rows are permitted; in practice the AssetCreationForm prevents more than one file being added.

```
<table class="mf-table mf-file-upload-table">
   <thead>
      <tr><th>File Name</th><th>File Size</th></tr>
   </thead>
   <tbody>
      <tr>
         <td class="mf-file-name">{{ file name }}</td>
         <td class="mf-file-size">{{ file size }}</td>
      </tr>
   </tbody>
</table>
```

Component: GeneratedMetadataCollectionForm

This component, unlike other components listed here, still uses old-style GWT methods to generate its content, which results in un-semantic HTML using deeply nested tables to layout elements.

A source is not provided as the implementation plan is simply to use the existing form generation technique, providing the least amount of specific formatting or rendering information permissible.

Status: described by Daniel

Component: AssetList

A list of assets. This is displayed even if there are no assets to list; in that case, a `<ul>` tag is created containing zero `<li>` tags.

Includes a wrapper div to allow for more convenient styling.

Identical to CollectionList, apart from the class.

Status: drafted by Lewis

Components included

- ListedResult

Example source

```
<div class="mf-result-list">
    <ul class="mf-asset-list">
        {{ zero or more ListedResults }}
    </ul>
</div>
```

Component: CollectionList

A list of collections. This is displayed even if there are no collections to list; in that case, a `<ul>` tag is created containing zero `<li>` tags.

Includes a wrapper div to allow for more convenient styling.

Identical to AssetList apart from the class.

Status: drafted by Lewis

Components included

- ListedResult

Example source

```
<div class="mf-result-list">
    <ul class="mf-collection-list">
        {{ zero or more ListedResults }}
    </ul>
</div>
```

Component: ListedResult

A summary of an asset, used when a list of assets is displayed.

Status: drafted by Lewis

Components included

- AssetInfoBlock (for AssetList only)
- AssetThumbnail
- DownloadButton (for AssetList only)

Example source

```
<li class="mf-listed-result" role="article">
    {{ zero or one AssetThumbnail }}
    <h2 class="mf-result-title">
        {{ name of result }}
    </h2>
    <div class="mf-result-description">
        {{ description of result }}
    </div>
    <div class="mf-listed-asset-body">
        {{ zero or one AssetInfoBlock }}
        <div class="mf-result-metadata-list-wrapper">
            <ul class="mf-result-metadata-list">
                <li><section>
                    <h3 class="mf-metadata-field">{{ metadata name }}</h3>
                    <div class="mf-metadata-value">{{ metadata value }}</div>
                </section></li>
                {{ ..one li for each metadata property.. }}
            </ul>
        </div>
    </div>
</article></li>
```

Component: AssetInfoBlock

A data island tag containing information about an asset, in JSON format. Used to make asset details available to JavaScript, and generally invisible to the end user.

Status: drafted by Lewis

Example source

```
<script class="mf-asset-info-block" type="application/json">
{
    “title": “{{ title of asset }}",
    “id": “{{ id of asset }}",
    “url": “{{ url that returns the asset }}",
    “metadata": [
        “...": “...",
        “...": “..."
    ]
}
</script>
```

Component: AssetThumbnail

An image thumbnail for an asset.

Status: drafted by Lewis

Example source

```
<div class="mf-result-thumbnail-wrapper">
    <img class="mf-result-thumbnail" src="..." />
</div>
```

Behaviour

Page behaviour is controlled with javascript. The portal designer can supply js code that will be delivered to the end user's browser along with the necessary GWT framework code.

Since the page is assembled in-browser by GWT, portal-designer JS needs to wait for the GWT page assembly to be complete before performing the designer's tasks. These hooks are used to notify designer-code about the completion of rendering events.

> *Implementation note*
>
> To create events, GWT needs to do something like this at the start of the browser window session:

```
// Create the events
var gwtOnPageLoad = document.createEvent('Event');
var gwtPageRendered = document.createEvent('Event');
var gwtDialogRendered = document.createEvent('Event');
var gwtDialogClosed = document.createEvent('Event');

// Define the event names, and that the events
// don't bubble and aren't cancellable.
gwtOnPageLoad.initEvent('gwtOnPageLoad', false, false);
gwtPageRendered.initEvent('gwtPageRendered', false, false);
gwtDialogRendered.initEvent('gwtDialogRendered', false, false);
gwtDialogClosed.initEvent('gwtDialogClosed', false, false);
```

`gwtOnPageLoad` event

This event will be triggered on the body tag once all the GWT javascript is loaded, but before any page elements are rendered.

Portal designer code will do something like this:

```
document.body.addEventListener('gwtOnPageLoad', function(e) {
    //
    // in here, e is the target of the event -- the body element
    //
}, false);
```

> *Implementation note*
>
> To trigger this event GWT needs to do something like:

```
document.body.dispatchEvent(gwtOnPageLoad);
```

`gwtPageRendered` event

This event will be triggered **on the body tag** immediately after the "page" is rendered or re-rendered. This will occur once when the HTML page is first fetched from the server and the default page has been presented; then again each time the user navigates between static pages and/or collection list pages.

GWT assembles the page via JavaScript, so portal author code that relies on the existence of DOM elements should run in response to this event to ensure DOM structures have been created.

Portal designer code will do something like this:

```
document.body.addEventListener('gwtPageRendered', function(e) {
    //
    // in here, e is the target of the event -- the body element
    //
}, false);
```

Implementation note

To trigger this event GWT needs to do something like:

```
document.body.dispatchEvent(gwtPageRendered);
```

`gwtDialogRendered` event

The portal interface sometimes renders modal dialogs that overlay the usual portal page, for example when displaying the New Asset dialog, or for playback of a media asset. Once a dialog has been created and displayed by GWT, this event is triggered **on the body tag**. To discover the dialog itself, portal author code can find the `div.mf-modal-wrapper` and inspect the contents.

Portal designer code will do something like this:

```
document.body.addEventListener('gwtDialogRendered', function(e) {
    var dialog = document.getElementsByClassName('mf-modal-container')[0];
    //
    // dialog is now (subject to error checking) the dialog container
    //
}, false);
```

Implementation note

To trigger this event GWT needs to do something like:

```
document.body.dispatchEvent(gwtDialogRendered);
```

`gwtDialogClosed` event

This is the event marking that a dialog previously rendered has been closed. This event is triggered **on the body tag**. Use this event to clean up after code executed in the `gwtDialogRendered` event.

Portal designer code will do something like this:

```
document.body.addEventListener('gwtDialogClosed', function(e) {
    //
    // in here, e is the target of the event -- the body element
    //
}, false);
```

> *Implementation note*
>
> To trigger this event GWT needs to do something like:
>
> ```
> document.body.dispatchEvent(gwtDialogClosed);
> ```

Appendix

Moved the out-of-scope/deprecated components to here for ease of browsing.

Component: FormTextItem (out of scope)

Note: this component is an aspirational one that is not planned to be implemented. It is preseved here for potential future use.

A form item used to collect a user's text input, using a `<input type="text">` tag.

AssetCreationForms will always include a FormTextItem/TitleItem to supply the asset's name, which adds an additional class to the outermost fieldset wrapper.

Some FormTextItems are compulsory (FormTextItem/RequiredItem) and this is represented by an additional class to the outermost fieldset wrapper.

Status: drafted by Daniel

Example source: ordinary FormTextItem

```
<fieldset class="mf-form-item mf-text-item">
   <label class="mf-form-item-label-wrap">
      <span class="mf-form-item-prompt">{{ item name }}</span>
      <input class="mf-field mf-text-input" type="text">
   </label>
</fieldset>
```

Example source: FormTextItem/TitleItem

```
<fieldset class="mf-form-item mf-text-item mf-title-item">
   ...
```

Example source: FormTextItem/RequiredItem

```
<fieldset class="mf-form-item mf-text-item mf-required-item">
   ...
```

Example source: FormTextItem/TitleItem/RequiredItem

```
<fieldset class="mf-form-item mf-text-item mf-title-item mf-required-item">
   ...
```

Component: FormSelectionItem (out of scope)

Note: this component is an aspirational one that is not planned to be implemented. It is preseved here for potential future use.

A form item used to collect a user's input selecting one of a range of specified options, which is a complicated way to say that this uses a `<select>` tag.

Some FormSelectionItems are compulsory (FormSelectionItem/RequiredItem) and this is represented by an additional class to the outermost fieldset wrapper.

Status: drafted by Daniel

Example source: FormSelectionItem

```
<fieldset class="mf-form-item mf-select-item">

   <label class="mf-form-item-label-wrap">
      <span class="mf-form-item-prompt">{{ name of field }}</span>
      <select class="mf-field mf-select">
         <option></option>
         <option>{{ first option }}</option>
         <option>{{ second option }}</option>
         <option>{{ ... }}</option>
      </select>
   </label>
</fieldset>
```

Example source: FormSelectionItem/RequiredItem

```
<fieldset class="mf-form-item mf-select-item mf-required-item">
   ...
```