Icons

A lot of DomUI components accept icons. DomUI supports multiple kinds of icons:

- Images represented by the **img** tag, with a source either somewhere from the webapp or from the Java code (as a Java resource). The component class for an image based icon is an **ImgIcon**.
- Font icons like FontAwesome (4.7 and 5.0 free currently supported), rendered as a span with the required css classes. The component class is **FontIcon.**
- SVG icons using an svg file from the webapp or from Java code (java resource), rendered as inline SVG inside a special span. The component class is **SvgIcon**.

The component classes are actual DomUI components, and as such they can be added to pages, components and whatnot whenever an icon is needed. But specifying an icon for a component is better done with something else as a component because a component can be used only once.. If you need the same icon multiple times it becomes necessary to specify the icon to use in another way, that allows multiple icon components to be made for the same icon.

DomUI 1.0 exclusively used Strings containing image files as icon specification in all components. That worked well for image based icons but failed badly for font icons. In addition having all those strings all over the place makes it really hard to see what icons are used. These are the reasons why 2.0 does it differently. 1.0 applications will need to be manually changed; usually adding an Icon.of(xxx) for icon xxx will suffice.

Specifying icons in components

To refer to an icon we use an **IIconRef**. An IIconRef is an interface which has only a few methods:

- NodeBase createNode() will create the DomUI component for the specific icon. The icon component created depends on the type of icon ref being passed, i.e. this creates either an SvgIcon, an ImgIcon or a FontIcon depending on what kind of icon is being added.
- css(String... classes) can be specified to add custom css classes to the icon specification. These will be added to the DomUI component once it is created. This allows you to alter the default look of an icon (which is important for font icons and svg icons), for instance like this:

```
DefaultButton save = new DefaultButton("save", FaIcon.faDiskO.css("is-primary"), a -> save());
```

In this example the icon specification does not return a component but an icon ref containing all of the information needed to create the icon component as soon as the button itself is created.

Making icons for file resources

For file based icons like image icons and svg icons it is possible to create an icon by using a path as follows:

```
IIconRef icon = Icon.of("img/btnSave.png");
```

The Icon.of method uses the extension in the path passed to determine the icon type. It returns an SVG icon ref for everything that ends with .svg, and an image icon ref for the usual extensions like png, gif, jpg.

While Icon.of can be quite useful it is better to define your icons somewhere in an enum class, where each enum implements IIconRef and represents a given icon. This allows you to easily see where you use what icons, and it allows you to easily change icons. An example implementation of this can be seen in the Theme enum which contains all (well, it should) of DomUI's default icons.

Font icons

Icon.of cannot be used for font icons because these do not have a path: a font icon is typically represented by a set of css classes where a given class selects a given css element containing both the font spec and the character in that font that represents the icon used. So for font icons we need something else as Icon.of.

Font icons however *can* be represented by an IIconRef as usual because these do not specify how the icon is created. To see how we would work with font icons let's look at FontIcon, the DomUI component that renders a font icon. This class only expects one or more CSS classes, and it uses these classes to select the font. To actually render a font icon you also need to make sure that the appropriate css file has been included in the header of the page (usually done by adding a HeaderContributor to the DomApplication.initialize() method).

To refer to individual icons in a font's set we have to use the css class names, but this is error prone. A better way is to again define a specific Enum class that represents all of the icons in the font set you use, and then to use those enum values (each implementing IIconRef) as the source for the class name. This again makes icons fully typed and compile time safe as any spelling error in a font icon's name does not compile.

Examples of using icon fonts can be found in DomUI's integration with Font Awesome.

FontAwesome integration

DomUI has two separate modules/jars for FontAwesome integration:

- fontawesome4 implements the icons for Font Awesome 4.7
- fontawesome5free implements the icons for Font Awesome 5, the free version.

To use either you must include either the one or the other, **not** both in your project (by adding the dependency in your maven pom or build spec). Currently DomUI **requires** you to use at least one of them as some DomUI default icons come from these.

After including the appropriate project you will have a new enum called FaIcon which contains an icon label for every icon in the appropriate FontAwesome package. So to add a button using a FontAwesome icon you could do:

```
new DefaultButton("yes", FaIcon.faCheck, a -> bleh());
```

As usual you can add css classes too:

```
new DefaultButton("yes", FaIcon.faCheck.css("is-danger"), a -> bleh());
```

which will get you a red icon (with the default DomUI theme).

When you use the FontAwesome integrations from DomUI you do not need to add the FontAwesome css files; these are automatically added. The mechanism for this is as follows:

- The module adds a services file for an IApplicationInitializer. This registers a class to be called when DomUI initializes, and this class takes care of adding the required HeaderContributor to the application so that it gets added to every page.

Adding other font packages

The fontawesome modules are good examples of what to do to add other font packages. They also contain a useful helper class (IconFromCss) to scan a css file and extract all font icon names from the set which are then rendered as an enum class. You can copy and adapt that class inside your own font module to generate the required enum.

Standard icons and remapping

The Icon class (part of DomUI) has a standard set of icons that is mostly based from the set provided by Font Awesome 4.7. It contains all of that set's icon names, and if you choose the fontawesome4 module these are normally the icons shown for these icons. But the Icon class has a "replacement" mechanism: you can change the actual icon used for any of the icons specified there by specifying some other icon. That other icon does not need to be a font icon: it can be any icon type.

This remapping is also used when you use fontawesome5free: that font happily renamed a lot of icons because otherwise life would be easy, and nothing is more fun than hunting down loads of strings in an application. But the FontAwesome 5 module handles the remapping of the Icon icons automatically and replaces the "old" icon name as used by Icon by the new representation inside Font Awesome 5.

The Icon class is for a standard icon set. The Theme class, already mentioned, should be used for default icons for common actions. This class, like Icon, has a remapping method.

One disadvantage of the remapping though is that it is global: you cannot have mapping per page.