DomUI component rules

When we started DomUI we did our best to have some rules for components. But of course we learned a lot, and in the best way possible: by making mistakes. The sad news, of course, is that those mistakes live on in the framework. This is why we have multiple versions of components, like LookupInput and LookupInput2. The higher the number the better the control is supposed to be (and incidentally the more stubbornly wrong we were).

If a component has no numbered version this does not necessarily mean it's perfect, though

Related to the quality of the components is the quality of the style sheets and style sheet rules associated with the components. DomUI has a highly flexible theming engine which allows for many ways of organizing style sheets. DomUI's styling also preceded the current flock of CSS preprocessors like Sass and Less, so it implements some logic by itself in the older stylesheets. Because DomUI was primarily used by a corporation with special UI needs there are multiple style sheets, most of them of questionable overall quality.

Starting with DomUI 2.0 all new components will get their styles from the SCSS stylesheet provider. These stylesheets use standard SCSS, and obey stricter rules around how components and pages should be styled. The existing "legacy" stylesheets should continue to work just fine, and we're taking care to not change the components in such a way that they are impossible to use with the older stylesheets. But new components and layout fixes will mostly be done in the new SCSS theme, "winter".

Stylesheet rules

Reset scripts - no longer used

The earlier DomUI stylesheets used a "CSS reset script" to clear the default styling off all HTML tags. The idea was that after this the behavior of the tags would be more easily controlled, and that this behavior would be the same for all browsers.

But using a reset sheet has quite a few side effects... Most of these are not too bad for websites and not too big applications, but for bigger stuff they get in the way:

- The reset script is not that easy to do well. So its effects percolate through to your own styles, often in unexpected results.
- Lots of functions actually want to *show* HTML. And those functions expect that html to be rendered in a reasonable way. Removing all formatting means that this does not hold, at all: by default it looks like flat text. The solution means adding back all those styles carefully removed by the reset script, sigh. Which means you STILL have to consider the "native" styles of tags.

Consequently the newer DomUI stylesheets do away with reset scripts completely.

The effects of this are not too bad: using the per-component classes (see below) you can more easily address styling of html tags used in the components and reset them only where needed. In addition DomUI mostly uses the styleless div and span tags, and just adds styling by giving those a class where needed.

The box model used is border-box.

The default (original) box model used by browsers leaves a lot to be desired. When you specify something like:

```
height: 100px;
width: 200px;
padding: 10px;
border: 2px solid black;
```

the *actual width of the element* is padding + border + width. The same applies to height: padding + border + height. This makes defining the actual size of element very hard because adding a border means decreasing the size if the element's size needs to be constant. It also causes huge problems with things like "width: 100%" because those cannot have border nor padding: it would make the element exceed the 100% width causing those annoying scroll bars.

There is a better box model that is currently well supported by all major browsers (and IE up to version 8): the *border-box model*. This model uses the defined width and height of an element as the actual rendered size, and subtracts from that size the sizes of borders and padding to get the content area's size. In effect this means that the above definition would *always* result in an element of *exactly* 200x100 pixels wide. The content area (inside the element) would be exactly 176x76 pixels big: 200 - 2*2 - 2*2 for width and the similar calculation for height.

The script used for this is in _core.scss, and applies the following:

```
html {
  box-sizing: border-box;
}
*, *:before, *:after {
  box-sizing: inherit;
}
```

This sets all tags to use border-box sizing by default, and makes it inherit so that changes made locally (should) percolate through.

See the following links for more information:

- Box sizing layout secrets

Browser support and tweaks

In those times of yore we had browsers and Internet Explorer. The latter was, well, evil. Supporting IE versions below 8 was what Hell probably looks like. Luckily a lot has changed: while IE is still a royal pain in the backside (you won't believe the trouble it has with the simplest stuff still) its rendering, at least, has become way better.

Now if only they would not LIE about the css styles in the developer tools....

For IE we try to do our best to make it work with the jokes implanted in the actual browser versions. We do not usually spend time getting something to work in the IE7 emulation mode in IE 11 or nonsense like that. The best way to get things to Just Work is actually to write them on Chrome or Firefox, and once they are working to try to see what IE makes of it. The other way is more problematic, usually.

Supporting old browsers (well, IE) is just too much work. So we focus on current browsers (meaning those that were current a year ago) to create style sheets, and we remove stuff in style sheets that were needed for Netscape 1.1.

If tweaks are needed please try to use feature discovery, and try not to depend on browser identification.

Older versions of the DomUI style sheets allowed embedded "preprocessing" Javascript, and knew about the browser version they were rendered for. This allowed per-browser version tweaks by conditionally including css. The SCSS version of the sheets do not support this and do not know the browser type.

One fragment per component

Each component must have its own stylesheet fragment which contains most of the scss needed to render the component. For a component like LookupInput2 there is a fragment called _lookupinput2.scss. These fragments are included in the master stylesheet called theme.style.scss. This same rule applies to the legacy stylesheets; these fragments are called like lookupinput2.frag.css.

When a new fragment is added you need (for now) to copy the style.theme.scss file and add the includes for the new fragments. In a later version the fragments will be located and included automatically.

Styling a component

Each component must have an unique CSS class "base name". These names usually start with "ui-" followed by a 3 to 5 letter abbreviation for the component's name. For example: the LookupInput component uses *ui-lui* as its base name. All classes added to html nodes should always starts with this css base name. Using the base name prevents name clashes when different components define the same class name, leading to oddities in rendering.

The root node for the component *must* define itself to have as its class the CSS base name. This ensures that all components can always be addressed by a CSS class - even when there is no apparent need, yet. The root node may have multiple classes, as long as:

1. They all start with the CSS base name (or ARE the CSS base name)
2. The CSS base name is ALWAYS present.

An example css structure for a simple DomUI button component could look like this:

```
<button class='ui-gbtn'>
  <span class='ui-gbtn-icon'></span>
  <span class='ui-gbtn-label'>He<span class='ui-gbtn-accel'>l</span>lo<
/span>
</button
```

All of the styles for a component *must* be assigned using CSS. Components should never contain code that hard sets the css attributes for the component. The only exception to this rule is when there is no other way to effect what is needed. Adding CSS related code inside the Java code makes the UI very hard to maintain.

Sharing styles

Components should **never** use classes from other components to handle their styling!! That would lead to very hard to maintain code: changing the style for one component now suddenly has an effect on some other component, and the styles were not designed for that in the first place!

You can define shared styles, but these should be clearly marked as such and be present in a separate stylesheet fragment. Shared styles have no CSS base name suffix.

Shared styles should normally only be used for small things, because sharing too much means that it becomes hard to change components using that style.

Using CSS selectors for the "parts" of a component

More complex components can consists of multiple html tags. For instance the DataTable component contains a table, a tbody, tr's, td's and more. The "root" of the component, as said, **must** have the CSS base class name which for DataTable is "ui-dt".

A common mistake is to now address the parts in the DataTable with selectors like:

```
.ui-dt td {...
```

This finds the td's in the DataTable allright. But the problem is: *it also finds all td's of tables that are INSIDE a DataTable cell!*

So a style that is supposed to work for a cell of the DataTable only now also applies for something *inside* such a cell. That is a bad idea.

So the rule is: **use only class names in selectors, do not use tags**.

Layout rules

One of the biggest design flaws in the earlier DomUI style sheets have to do with layout. So in the new style sheets we attempt to do better, by defining stricter rules. The disadvantage of the new rules are that sometimes more css is needed to style a page, or that extra "layout" components are needed to get something to look good. But the advantage should be less odd layout cases and easier CSS.

Two basic types of components

For the purpose of the discussion we divide the components in two big "groups" related to their layout:

- The "inline" component group. These components use limited width, and often occur together with other components in some "inline" way. Good examples of inline components are Text<T>, LookupInput, CheckBox, ComboLookup and all.
- The "panel" component group. These usually contain other components and show as a block. They can either be complex components like DataTable and Tree, or they can be Panels or Headers.

Inline component rules

Component structure

Inline components should, by default, all behave as an inline-block. So by default it should be possible to place inline components one after another.

All inline components must have a single Node that contains all of the component. This is important because it allows the parts of the component to be addressed relative to its container, so that alignment rules can be applied. So a proper component would be:

```
<div class='ui-myc'>
  <input type='email' size='12'>
  <button class='ui-myc-btn'><span class='fa-icon' /></button>
</div>
```

while a bad, bad one (hello DateInput) would be:

```
<div class='ui-myc'>
  <input type='email' size='12'>
</div>
<button class='ui-myc-btn'><span class='fa-icon' /></button>
```

The latter was used in some older DomUI components- and styling them is also called the seventh circle of Hell.

Inline component layout

By default an inline component should **not** have any padding around it. So adding two inline components after each other should show with those components touching each other! It is the responsibility of the *container* to place components in such a way that groups of components "look nice".

The reason for this rule is that having paddings around the components makes it hard to construct *other* components from existing ones.

Take the following example:

Say that the SmallButton itself came with 5px padding all around it. The Text<T> component also has this. But a common "supercomponent" is to combine a Text with one or more small buttons, and for that it is customary to have the buttons touching the Text control and each other:

[IMAGE HERE]

To get this done the new component now has to "undo" the styles of the components, then add its own. That is error prone.

This rule also means that adding "naked" components together, without help, will look like shit. This btw has always been the case, but it was just a different kind of the smelly stuff.

Forms

Input components are normally used inside forms. A form in DomUI is not a component but is built by a *FormBuilder*, a special class which helps with creating a nice layout for a form. The result of a FormBuilder is a set of Nodes with special styles that together with the components *should* force them to look nice.

The current "best version" of a FormBuilder is the FormBuilder from package form4. This formbuilder allows building both vertical (default) and horizontal forms, and uses data binding extensively.

Vertical forms for form4

A vertical form is built using a table, with each row containing one cell for the label and one for the control. The details can be found here.