API Options

Rich Text API

This document describes different approaches that can be used by javafx to offer Rich Text support.

The two main options described here are:

Attributed String

In all likelihood, in this style of API the existent Text node would be modified to allow style objects to be applied to ranges of its text content. This design is used by several text layout solution (CoreText, Pango, DirectWrite, SWT, AWT, etc)

DOM

This option involves adding new nodes to the scene layer, in particular a new parent node with the capability of grouping several text spans (Text node) and other graphic elements (ImageView, Canvas) and laying them out in a text paragraph following all applicable text layout rules to the entire content (bidi reordering, wrapping, justification, etc). This option is similar to HTML+CSS.

Whichever design is chosen, the following statement should always be true:

> *Any style that can be applied to text layout using code should also be possible to be described in CSS.*
> *The usage of CSS should not be limited to just the simple case (className/id). It should, as much as possible, allow for any css selectors (including those based on node relationship).*

For reference, the following are the different ways a node can be styled in javafx:

Adding class name	`node.getStyleClass().add("styleClass");`
Setting id	`node.setId("text1");`
Setting inline style	`node.setStyle("-fx-font: 24pt serif;");`
Setting property directly	`node.setFont(new Font("serif", 24));`

Use cases

This section presents a few different use cases with a possible implementation and discussion using the options described above.

Simple case, make one word bold

Sample text: Hello **Bold** World.

AttributedString

```
Text text = new Text("Hello Bold World");
TextStyle style = new TextStyle();
Font font = text.getFont();
Font boldFont = Font.font(font.getFamily(), FontWeight.BOLD, font.getSize();
style.setFont(boldFont);
text.setStyle(6, 10, style);
root.getChildren().add(text);
```

CSS AttributedString

```
Text text = new Text("Hello Bold World");
TextStyle style = new TextStyle();
style.getStyleClass().add("bold");
text.setStyle(6, 10, style);
root.getChildren().add(text);
scene.getStylesheets().add("file://mysheet.css");
```

mysheet.css:

```
.bold {
  -fx-font-weight: bold;
}
```

New APIs added:

```
new method
Text#setStyle(
int start - inclusive start offset
int end - exclusive end offset
TextStyle style - the style to be applied to the range.
)
new class TextStyle
```

Notes:

- Text#setStyle merges new style with existent styles (by overwritten them)
- start,end could be moved to TextStyle but that would make TextStyle not reusable
- TextStyle: does it need to support all the styling methods available to node (see #table)?
    - does it need TextStyle.setFont(), setUnderline(), setWhatever() ?
    - does it need TextStyle.setStyle() (in the same form as Node.setStyle()) ?

An alternative is a dictionary style API:
For example:
`TextStyle.addStyle("-fx-font", boldFont);`
or
`TextStyle.addStyle("-fx-font", "12pt bold system");`

This is similar to other system:
For example, Apple CoreText a CFAttributedString is a combination of the CFDictionary and a CFString.
HTML also works this way, the same in javascript would be written as:
`node.style.FxFont = "12pt bold System";`
or
`node.style["FxFont"] = "12pt bold System";`

(note that in javascript any object can be a hashtable, thus the above is just adding the key-value pair to the hashtable style).

In addition, the option to set a class name and id should for a TextStyle should be available:
`TextStyle#getStyleClass().add(String className);`
`TextStyle#setId(String id);`

DOM

```
Paragraph paragraph = new Paragraph();
Text text0 = new Text("Hello ");
Text text1 = new Text("Bold");
Text text2 = new Text(" World");
Font font = paragraph.getFont();
Font boldFont = Font.font(font.getFamily(), FontWeight.BOLD, font.getSize();
text1.setFont(boldFont);
paragraph.getChildren().addAll(text0, text1, text2);
root.getChildren().add(paragraph);
```

CSS DOM

```
Paragraph paragraph = new Paragraph();
Text text0 = new Text("Hello ");
Text text1 = new Text("Bold");
Text text2 = new Text(" World");
text1.getStyleClass().add("bold");
paragraph.getChildren().addAll(text0, text1, text2);
root.getChildren().add(paragraph);
scene.getStylesheets().add("file://mysheet.css");
```

Embed image in a text

Sample text: Hello  World.

AttributedString

```
Text text = new Text("Hello \uFFFC World");
Image img = new Image("file:///myimage.png");
Font font = text.getFont();
FontMetrics fm = Toolkit.getToolkit().getFontLoader().getFontMetrics(font);
GlyphMetrics gm = new GlyphMetrics()
gm.setDescent(fm.getDescent());
gm.setAscent(img.getHeight() - fm.getDescent());
gm.setWidth(img.getWidth());
TextStyle style = new TextStyle();
style.setGlyphMetrics(gm);
text.setStyle(6, 7, style);
ImageView imgView = new ImageView(img);
root.getChildren().addAll(text, imgView);
//every time a layout happens
Path p = new Path();
p.getElements().addAll(text.impl_getRangeShape(6, 7));
p.setLayoutX(text.getLayoutX());
p.setLayoutY(text.getLayoutY());
Bounds bounds = p.getBoundsInLocal();
double x = bounds.getMinX() + text.getLayoutX();
double y = bounds.getMinY() + text.getLayoutY();
imgView.setLayoutX(x);
imgView.setLayoutY(y);
```

Notes:

- This case can't be expressed using CSS with an Attributed String API
- Image is vertically aligned at the bottom of the text layout

DOM

```
Paragraph paragraph = new Paragraph();
Text text0 = new Text("Hello ");
ImageView imgView = new ImageView();
imgView.getStyleClass().add("imgView1");
Text text2 = new Text(" World");
paragraph.getChildren().addAll(text0, imgView, text2);
root.getChildren().add(paragraph);
scene.getStylesheets().add("file://mysheet2.css");
```

mysheet2.css:

```
.imgView1 {
  -fx-image: "file:///myimage.png";
  vertical-align: bottom; //Currently not supported on FX
}
```

Use CSS selectors to implement theming

Suppose we are implementing a code editor, code has different token (keywords, literal, comments), each token as a style (described in the stylesheet). The editor has to support theme, which means that at anytime a new theme can be set and the style for the tokens replaced.

Sample text: public String name = "Joe";
In HTML, that can be display with:

```
<div id="editor" class="eclipse">
        <div>
                <span class="keyword">public</span>
                <span class="whitespace"> </span>
                <span class="normal">String</span>
                <span class="whitespace"> </span>
                <span class="field">name</span>
                <span class="whitespace"> </span>
                <span class="symbol">=</span>
                <span class="whitespace"> </span>
                <span class="string">"Joe"</span>
                <span class="symbol">=</span>
        </div>
</div>
```

There is eclipse.css with the content:

```
.eclipse {
        -fx-font: 12pt Monaco; //this is the default font for the editor (and all its children)
}

.eclipse .keyword {
        -fx-color: "purple";
        -fx-font-weight: "bold";
}

.eclipse .string {
        -fx-color: "blue";
}

/* etc */
```

The way this works is, for example, if there is node with the "keyword" class which is a descent of a node with the "eclipse" class then apply the purple bold style.

Now assume that a new netbeans.css is added:

```
.netbeans {
        -fx-font: 13pt fixed;
}

.netbeans .keyword {
        -fx-color: "blue";
}

.netbeans .string {
        -fx-color: "orange";
}

/* etc */
```

To change, at any time, the theme of the editor from "eclipse" to "netbeans" all that needs to be done is change the class name of "editor" from "eclipse" to "netbeans", in javascript:
`document.getElementById("editor").className = "netbeans";`

and everything works.

Note that the eclipse.css does not need to be removed. In fact, all stylesheets can be loaded at startup time (which is usual in HTML).

The span node with content equals to "public" always has the "keyword" class name, but it is no longer a descent of a node with the class name of "eclipse", which means the purple bold style does not apply to it anymore. It is now a descent of a node with the class name "netbeans" and as such it will have the style blue applied to it.

With a DOM Style API this type of situation can be solved the same way it is done in HTML.
With an Attributed String API this will have to be done by hand, adding and removing styles one by one everywhere in the scene graph.