Printing for JavaFX

JavaFX8.0 intends to include printing support for the first time.

The JIRA for this feature is http://javafx-jira.kenai.com/browse/RT-17383

Please post comments on the following there.

I have posted a draft core printing API proposal at http://cr.openjdk.java.net/~prr/fxprinting/

Its the full 8.0 javadoc but the only relevant packages are
javafx.print
javafx.print.attribute

It provides the core functionality to do
 - printer discovery
 - job creation and configuration
 - user interaction
 - imaging scene graph nodes to the printer

1) PrinterJob - the class that controls the printing process and provides support for
 * print and page setup dialogs which will be the platform native dialog
    at least on Windows and Mac OS X.
* obtaining job settings (via the delegate JobSettings class)
* setting the printer for a job
* printing pages containing Nodes (more later)
* print job life cycle/status

2) Printer - the class returns an immutable object that encapsulates a native printer.
 * provides API to find the default printer
 * provides API to find all installed printers
 * provides API to retrieve printer defaults and supported values via the
   delegate PrinterAttributes class

3) PrinterAttributes provides access to the configuration
  of the printer such as the supported paper sizes, and the
   default paper size.
  Most of these attributes are defined as classes or enums in
  the javafx.print.attribute subpackage
  They map to IPP print attributes as used by javax.print,
   .NET printing and CUPS/OS X printing.

4) A JobSettings is the current settings of the above attributes
    for a job, plus others such as "job name" that aren't printer attributes.

5) PageLayout is made up of paper size, orientation and margins
    and results in the area of the paper to be printed.

The basic unit of printing is a Node. This can be a parent/group node.
Pagination of large amounts of content into separate nodes per page
is up to the application. That is the only support in the API here.

Simple Example

```
import javafx.scene.shape.Circle;
import javafx.print.PrinterJob;
...
Circle node = new Circle(100, 200, 200);
PrinterJob job = PrinterJob.getPrinterJob();
if (job != null) {
   if (job.printDialog(null)) {
      job.printPage(node); job.endJob();
   }
}
```

What's missing ?

Likely I am not remembering off hand all the things I should so treat
this as a non-exhaustive list. Also these may or may not ever make it.
Plans/ideas can be superseded by better ones, or may just be too much
for a release.

At some point it might be useful to provide more direct scenegraph
support for something like a node iterator that splits content into
separate nodes which fit available space.

Consideration is also being given to providing a way to treat the page directly as a javafx.scene.canvas.Canvas, so that an app can render using immediate mode APIs too. This is still just an idea.

Webview need to support printing its content somehow since the app can't correctly format that content itself.

A default CSS style applied for printing of UI controls that's more printer friendly. Caspian uses too much grey for the printer.

Headers, footers, watermarks may be interesting but if they aren't widely useful without too many "knobs" it may be best left to the app

"Content" based printing where the app supplies an image or text or URL and we format it for the printer. The main reason this might be interesting that I can see is in the future for mobile devices which may have minimal printing support which is content oriented like this. In all likelihood this feature does not make sense to add until we have a need and something to actually try it out on.

What's probably out of scope ?

Trying to automatically relay out an application window with scrollbars and columns of content etc is.

Rendering of large tables etc is not best handled by rendering a TableView directly. A reporting package might be a better approach.

Pros and cons of a call back model :

In this API proposal an application directly prints a page one time by providing the scene graph content to print. So it has direct control over printing process whereas printing in JDK's Java 2D API requires that you implement a call back interface and provide this to the Printer Job. This has the advantage that once printing begins its under the control of the implementation so we don't to worry about partially completed print jobs being leaked by the application and the ensuing question as to when these can be cleaned up.

For an immediate mode API like Java 2D it seems more natural to a developer on to draw a page, throw it and be done. But some implementation issues mean that its likely that Java 2D needs to ask the page to render an arbitrary number of times, so having the call back ready to call as mean times as needed solves this.

On the other hand it can sometimes make things more difficult for the application since its not clear to the app when the implementation is done printing a particular page.

Some of the aforementioned implementation issues could affect FX printing but the scene graph is already effectively a call back interface. Its provided to the implementation which renders it as needed.
So a call back interface could be redundant although Canvas may be something of an exception.

Threading and updating a Node.

What does a developer needs to know about threading when adding printing to an application?

There are two main points to remember.

1) It will likely be recommended to invoke printing a page from some thread other than the FX thread as the printing implementation might need to take steps internally to prevent such a relatively long running activity from blocking the application's responsiveness. This is not a prohibition, just a recommendation.

2) When printing involves a node that is attached to a Scene,
as usual the node can only be updated from the FX user thread,
but additionally care needs to be taken that any update is
not during printing the node, else this would be a concurrent access.

Some background and detail :-

The goal is to provide for printing nodes that
are attached to a Scene, and nodes that are unattached.
Printing can be performed on any thread.

That is not to say that printing is multi-threaded from
a user perspective. Its single threaded, and the methods
include appropriate synchronized modifiers to ensure this,
but its not sensitive to the thread on which invocation of
the PrinterJob methods is performed.

Note the following normal FX rules stand :

1. Before adding a node to a Scene it may be constructed and modified on any thread
2. Once a node is added to a Scene all modifications must take on the FX thread
3. Concurrently modifying and accessing a node's state from different threads is
   not allowed under any circumstances, else the results are undefined.

In printing the additional rule is that

4. You cannot update a Node ***during the printing of that node to a page***.

Think of printing a page as temporarily attaching the node to a printer scene and it's being automatically
removed from that printer scene when the printPage(Node) method returns.

Once it has returned, whilst its probably most natural for an application to then update the
node on the same thread on which the printPage() method was executed but :-

* If the node is attached to an on-screen Scene, it must be updated only on the FX thread

* If the node is unattached, then with appropriate synchronisation the application
can choose to do the update on some other thread.