Introducing the Flexicious Flex Dashboard Platform

The **Flexicious Dashboard Platform** is a new product from Flexicious that empowers RIA developers to provide feature rich, expressive, highly customizable, and deeply engaging dashboard experience to their users.

It provides you with a lot of the base functionality that is standard in dashboard applications, in addition to a number of highly productive features that is sure to WOW your customers.

Right out of the box, you get:
A container for your dashboard that provides;

1. Support for configurable layout of dashlets, and provision for users to move and resize dashlets, integrated with preference persistence so dashboard loads exactly how the user left it.
2. Ability to organize dashlets in zones. Ability to drag and drop dashlets between zones, again with preference persistence mechanism.
3. Ability to Print Preview and Print the dashboard or a single dashlet
4. Ability to export the Dashboard to PDF
5. Ability for your users to save the current view of the dashboard via preferences
6. Ability Show/Hide Dashlets, control which dashlets appear where and at what size.
7. An auto refresh mechanism
8. Ability to control dashlet operations including:
    a. Move dashlets via a convenient drag and drop
    b. Resize dashlets
    c. Expand/Collapse Dashlets
    d. Open/Close Dashlets
    e. Maximize/Minimize Dashlets
    f. Loading Animation
9. Based on the Spark skinning model, provides the ability to completely skin the look and feel of the dashboard as well as the dashlets.
10. A number of professional, excellent looking built in skins

Check out the demo here
Read the getting started guide here.
Get the PDF copy of the Getting Started Guide here.
Download the trial here

Getting Started with the Flexicious Dashboard Framework.

The dashboard framework was built with a number of use cases in mind.

1) The first step is to download the trial project, which you can get from http://www.flexicious.com/resources/trials/DashboardTrial.zip
2) Extract it to your local drive, and open it in Flash Builder. The zip above is a full functional Flex project, the only missing piece being the Trail SWC which you get in the next step.
3) When you request a trial, you are sent a 30 day trial swc (DashboardTrial.swc), drop this swc in the libs folder of the project from the step above.
4) Right click on ComponentSample.mxml -> run as -> Web Application.

In the sample project, we demonstrate a few major concepts:

1) Struture of a typical dashboard: The Dashboard container contains one ore more Drag Drop Zone. If have seen the share point designer, this is a concept similar to that. So in the overall container, you can group dashlets on basis of their size and group them into one or more drag drop zones. As we will see shortly, you do not need to have multiple zones, in case you want a free flowing layout for your dashlets, you can simply have a single drag drop zone, and wrap them all in the same zone.
2) Layout of the dashboard zones: The main class for your dashboard is the com.flexicious.components.dashboard.DashboardContainer. This component indirectly extends from spark.components.SkinnableContainer, and thus, inherits all of the skinning capabilities introduced in Flex 4. As you will observe from the examples, this plays a big role in how the component works. The default skin for this component is the com.flexicious.skins.dashboard.DashboardSkin. This skin contains the top level title bar, and a group (the contentGroup) for the drag drop zones.
3) We have demonstrated 4 separate ways of laying out your dashlets in the samples. There can be more, but these should cover most scenarios:
    a. Drag Drop Zones with Vertical Layout Stacked horizontally side by side: Basically, in this scenario, we have three drag drop zones with widths set at 33% each, and height set at 100%. This setup prevents the appearance of the scrollbar, and the dashlets are individually positioned to occupy a set percentage of the zones available area. So the layout mechanism here is:
        i. Dashboard : Horizontal Layout
        ii. Drag Drop Zones: Vertical Layout
        iii. Please note, the mechanism to specify the Dashboard Layout is via the “contentLayoutFunction”. This should return a layout object. There is also the maximizedContentLayoutFunction, which is the layout used when the user maximizes a dashlet. The only

reason you would use this is to define padding and such in the maximize view. The drag drop zones however, define their layouts inline, as you can observe from the examples.

iv. In this type of a layout, you may want to set the "movable" and "resizable" flags of the dashlets to false. This is because the layout and positions of the dashlets are calculated by the layout you choose. If you enable these flags, the users positions clash with the layouts calculations and it results in a substandard experience. However, these layouts are a good choice for the "draggable" flag, where the user can drag and drop dashlets so dashlets that are important to the user can appear on top, and with the preference persistence mechanism, these positions are then remembered when the user loads the dashboard in the future.

b. Drag Drop Zones with Horizontal Layout Stacked Vertically : Similar to the example above, but the drag drop zones are stacked vertically here. So the layout mechanism here is:
    i. Dashboard : Vertical Layout
    ii. Drag Drop Zones: Horizontal Layout
c. Asymmetrical Drag Drop Zones: In this case, the dashboard simply has a basic layout. In this specific example, we have forced the zones to fit inside the visible area of the dashboard, so there is no need for a scroll bar. However, this is not a must. If you wish to add a scrollbar, please refer to the example below. You simply really need to wrap the contentGroup in a scroller. In this example, we have:
    i. Dashboard : Basic Layout
    ii. Drag Drop Zones: Layout defined per drag drop zone.
d. Single Drag Drop Zone, Basic Layout: In this example, we simply have one zone. The dashlets define x (or left), y (or top), width, height attributes. These are then used to render the dashboard. This example also demonstrates addition of scrollable content. The idea is simple. If the height of the dashlets and their X/Y co-ordinates make the drawn dashlets extend beyond the visual bounds of the dashboard containers view port, scroll bars will be shown. The scroller is added via a custom skin, as is demonstrated in the com.flexicious.sample.dashboard.customskins.bright.DashboardSkin
    i. Dashboard : Basic Layout
    ii. Drag Drop Zone: Basic Layout.
    iii. Tip: In this layout mechanism, you might wish to use the design view to define initial positions and sizes of your

dashlets. The way to do this would be to temporarily move your dashlets inside the dashboard tag for using the design view, and then moving them back into the only drag drop zone in mxml once you are done positioning your dashlets.

iv. In this type of a layout, you may want to set the "movable" and "resizable" flags of the dashlets to true. This is because a basic layout is similar to a canvas, where each component defines its own bounds. So the user is free to resize, move and drag dashlets around as they see fit. With the preference persistence mechanism, these positions are then remembered when the user loads the dashboard in the future. Since there is just one drag drop zone, it does not make sense to set the "draggable" flag, since there are no other zones to drag to, and the order of children dashlets does not matter from a positioning persepective.

4) Skinning/UI customization for the dashboard and dashlets: Since both the dashboard container as well as the dashlets extend from SkinnableComponent, you can leverage the excellent skinning mechanism that is a part of Flex 4. There are a few skins that you will be interested in:
    a. The Dashboard Skin: This is the skin responsible for drawing the top level dashboard. Its main parts are:
        i. Top Group
            1. Button Group: The group responsible for holding the buttons that you enable using the enable* flags on the dashboard container.
            2. Title Display: The label that shows the title.
        ii. Content Group: The container responsible for holding the drag drop zones that hold the individual dashlets.
        iii. Maximized Content Group: The group that is brought into view when the user maximizes an individual dashlet.
    b. The Dashlet Skin
        i. Top Group
            1. Button Group : Same as above
            2. Title Display: Same as above
        ii. Content Group: Same as above
    c. The Dashboard Button Skin
        i. There are specialized skins to only show the button icon image, along with the normal, down, and up states.
    d. The Dashboard Popup Button Skin
        i. This is a special skin for the Flexicious MultiSelectComboBox. (For those of you who did not notice, the MultiSelectComboBox is the same

component that is used as a filter in the Flexicious Grids). We are using it here as a UX for the user to control which dashlets are visible. Since we want the popup button to look like the other buttons, we have a specialized skin that is used to achieve this look and feel.

In the custom skins that ship as a part of the sample, we demonstrate the usage of these skin parts to provide a fully customizable look and feel to your dashboards.

5) API : As usual, the best way to understand any API is to see it in action. So first step would be to download the trial project and play with it. The second step would be to review the ASDocs. We are still polishing and improving the ASDocs, but most items that you will need to work with should be documented. The naming conventions are very similar to what you are used to with the current Flexicious API, in that there are numerous enable* flags on both the dashboard object and the dashlet object.