JIDE Dashboard Developer Guide

Contents

Purpose of This Document

> **dash · board** (dāsh'bôrd', –bōrd')
> n. A panel under the windshield of a vehicle, containing indicator dials, compartments, and sometimes control instruments.

The dashboard in a vehicle contains instruments and controls pertaining to operation of the vehicle. It is critical because it displays the important information in the real time and gives user the quick access to common tasks. *JIDE Dashboard* is as important except it is not for a vehicle but for a software application.

Imagining your application deals with real time data, there are tons of information you want to show to your users (such as a stock trading or a network device monitor application). There certainly isn't enough space to show all of the information. It's also hard to guess what users really want to see. So how about let user choose what to display? That's where you need *JIDE Dashboard*.

JIDE Dashboard provides a place holder to show all kinds of widgets so that user can perform common task and quickly access to important information. Instead of letting the developer design the layout and place the widgets, *JIDE Dashboard* allows your users to fully customize it. All you need to is to provide the widgets. Your end users will drag the widget from the widget palette and place them at the places they want.

It is worth noting that the dashboard concept is used widely in the computer industry. Mac OSX has Dashboard[1]. If you google "dashboard", the first entry is actually Mac OSX Dashboard. I am glad that they didn't trademark it. Microsoft Vista™ has Sidebar[2] and Yahoo! has Yahoo! Widgets[3]. All those three products are for the same purpose. Of course those three products are for the operation systems v.s. *JIDE Dashboard* is specifically designed for your application. On the other hand, Google's iGoogle™ [4] and Yahoo's My Yahoo![5] are web implementation of the same concept.

This developer guide is designed for developers who want to learn how to use *JIDE Dashboard* in their applications.

Features

Here are the main features of *JIDE Dashboard*.

- Dashboard supports drag and drop to arrange the widgets.
- Support built-in palette to list all available gadgets.
- Supports multiple dashboards
- Built-in persistence of the layout of all dashboards and widget settings.
- Supports any component as the widget.
- Supports column resizable
- Gadget component supports maximization/restoration

JIDE Dashboard

Here is what dashboard looks like under Window XP L&F with Office2003 style.

[1] http://www.apple.com/macosx/features/dashboard/ (Mac OSX Dashboard)

[2] http://vista.gallery.microsoft.com/vista/SideBar.aspx (Microsoft Vista Sidebar)

[3] http://widgets.yahoo.com/gallery/ (Yahoo! Widget)

[4] http://www.google.com/ig (Google's iGoogle)

[5] http://my.yahoo.com/ (Yahoo's My Yahoo!)

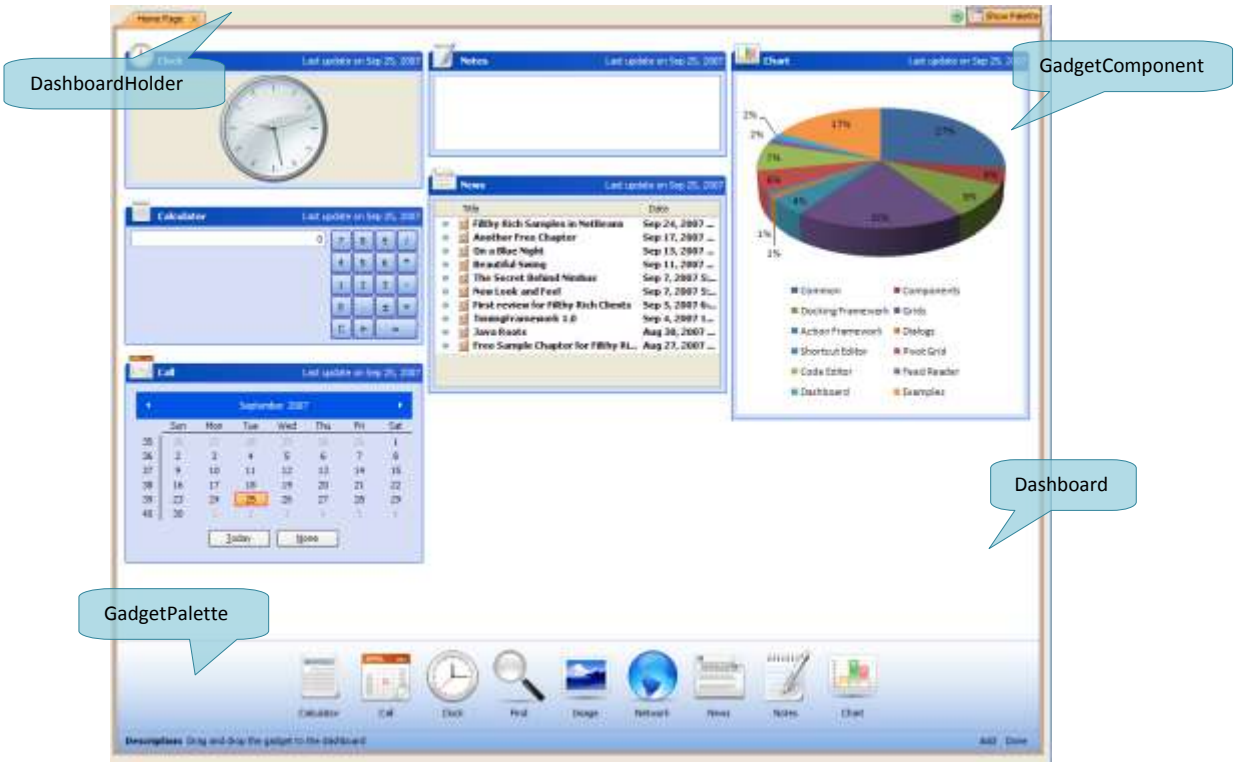


Figure 1 Overview of Dashboard

On the top level, there is *DashboardHolder* which holds one or more Dashboard. It also provides a place to put the *GadgetPalette*. The *Dashboard* contains *GadgetComponent* which is created by *Gadget*.

The gadgets' position can be rearranged using drag-n-drop. See below for an example. As you can see, we provided a nice transparent effect to guide you during the drag-n-drop process.

Figure 2 Drag-n-drop to rearrange gadgets

Overview of JIDE Dashboard APIs

There are five main classes in *JIDE Dashboard* APIs – *GadgetManager*, *DashboardHolder*, *Dashboard*, *Gadget*, and *GadgetComponent*. Except *GadgetManager*, we mentioned all others in the overview screenshot above.

You can view *GadgetManager* as the model of the whole *JIDE Dashboard*. It has a list of dashboards and gadgets.

Dashboard is the panel that contains all the gadget components. When a dashboard is created, it is empty. User will click the "Show Palette" button to show the palette and drag gadget from the palette onto the dashboard. Where does the palette find out all available gadgets? It is from *GadgetManager*. Developer will create all *Gadget*s and added them to *GadgetManager*. In addition to manage the list of gadgets, *GadgetManager* also manages a list of dashboards.

You may think *Gadget* is a *Component*. In fact, *Gadget* is just an interface containing the name, title, icon (small and large) etc information. We also have *AbstractGadget* that implements *Gadget* interface. It is not a *Component* either. When a gadget is dragged and dropped to the dashboard, we will call a create method on *Gadget* interface to create the actual component which is what we called *GadgetComponent*. *GadgetComponent* again is an interface but those who implement it must also be a real *Component*. When the component is removed from the dashboard, we will call the dispose method. Here are the two methods to create and dispose the *GadgetComponent*.

```
GadgetComponent createGadgetComponent();

void disposeGadgetComponent(GadgetComponent component);
```

By using this one-Gadget-multiple-GadgetComponent design, we are able to support two important features of dashboard:

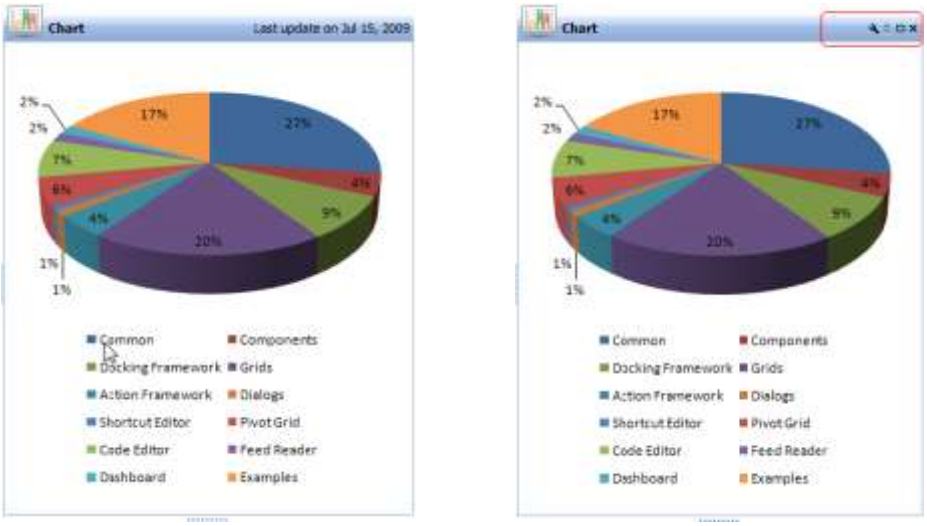
1. Allow the same gadget to be displayed several times on the dashboard (most likely with different settings).
2. Allow any component to be used as gadget component without extending a concrete class provided by us.

All gadget components created by *GadgetPalette* via drag-n-drop support further drag-n-drop by default. But if you call *createGadgetComponent* yourself to create a component and add it directly to *Dashboard*, it won't support drag-n-drop. It's not surprise as we don't have control over what component you will create in this case, so we won't get the chance to add the necessary mouse listener. In this case, you can call *GadgetManager#installListeners* method so that the component can be drag-n-dropped just like those created by *GadgetPalette*. For the same reason, if you remove it from dashboard and want to call *disposeGadgetComponent*, you should call *GadgetManager#uninstallListeners* to avoid any memory leaks.

GadgetComponent

Any component can be the *GadgetComponent*. We use *CollapsiblePane* as the default gadget component and implement a class called *CollapsiblePaneGadget*. *CollapsiblePane* supports collapsible feature, which is a useful feature for a gadget component.

In *CollapsiblePaneGadget*, the title buttons are not visible by default. Instead, you can set a message, which will be shown in the title pane area. As one of the purposes of dashboard is to display the information of your system, this message is a good place to show the current status of this gadget. When the mouse is moved over the title pane, the title buttons will be displayed. By default, we have three buttons – collapse/expand, maximize/restore and close. The fourth button as you see below is added in our demo to show you how to add extra buttons.



In your application, you can also create your own *GadgetComponent*. The source code of *CollapsiblePaneGadget* mentioned above is available in the demo. We also create *RegularPanelDashboardDemo* and *DockableFrameDashboardDemo* to show you how to create your own *GadgetComponent* using a regular JPanel and a *DockableFrame* respectively. Again, the source code for both demo and the gadgets are available.

Multiple Dashboards

JIDE Dashboard supports multiple dashboards. *GadgetManager* has all the necessary methods to support this feature. If it is up to you how you arrange the multiple dashboards. You can make a JList on the left to display the dashboard names and a JPanel with CardLayout to switch to different dashboard depending on the selection in the JList.

The tabbed pane is certainly a good way to display multiple panels. So we created *DashboardTabbedPane* which implements *DashboardHolder*. The *DashboardTabbedPane* uses *JideTabbedPane* to support multiple dashboards. See below.

The toolbar on the right side of the tabbed pane can be customized by overriding *createToolBarComponent* of *DashboardTabbedPane*. See blow for the code and the result.

```
DashboardTabbedPane tabbedPane = new DashboardTabbedPane(manager){
  protected Container createToolBarComponent() {
    Container toolBarComponent = super.createToolBarComponent();
    toolBarComponent.add(new JideButton("My Button"));
    return toolBarComponent;
  }
};
```

Persist the Layout

DashboardPersistenceUtils is the class that can save *DashboardHolder* as xml file and load them back. In addition, the gadget's application specific setting will be saved as long as the *GadgetComponent* correctly implements the *getSettings* and *setSettings* methods.

The reason we didn't build this feature into *DashboardHolder* itself but provide a separate class for it is because we think you might have your own to persist it as the xml format in your application.

All methods on *DashboardPersistenceUtils* are static. There are several overloaded *load()* methods, which loads dashboard layout and gadget settings from an xml file, or from InputStream, or from Document (org.w3c.dom.Document).

The save() method will save dashboard layout and gadget settings as Document or file. Please note *save()* method will use classes *XMLSerializer* and *OutputFormat*. Those two classes are part of xerces.jar before but now they are also part of JDK5. If you are using JDK5 and above, there is nothing you need to do. However if you are still using JDK1.4.2 or below, you need to include xerces.jar in your class path. Otherwise you will get *NoClassDefFoundError* during runtime.

Dashboard Events

GadgetManager will fire *DashboardEvent* and *GadgetEvent* when something happened. You can listen to the events by adding a *DashboardListener* or a *GadgetListener* to *GadgetManager*. You will receive an notification when

- A dashboard is added: DashboardEvent.DASHBOARD_ADDED
- A existing dashboard is removed: DashboardEvent.DASHBOARD_REMOVED
- A gadget is added: GadgetEvent.GADGET_ADDED

- A gadget is removed: GadgetEvent.GADGET_REMOVED
- A gadget is shown on a dashboard: GadgetEvent.GADGET_COMPONENT_SHOWN
- A gadget is hidden from the dashboard: GadgetEvent.GADGET_COMPONENT_HIDDEN
- A gadget component is created: GadgetEvent.GADGET_COMPONENT_CREATED
- A gadget component is disposed: GadgetEvent.GADGET_COMPONENT_DISPOSED
- A gadget component is maximized: GadgetEvent.GADGET_COMPONENT_MAXIMIZED
- A gadget component is restored: GadgetEvent.GADGET_COMPONENT_RESTORED
- A gadget component is resized: GadgetEvent.GADGET_COMPONENT_RESIZED

You can choose to handle those events by looking at the event id.

Internationalization Support

All Strings used in *JIDE Dashboard* are contained in one properties file called dashboard.properties under com/jidesoft/dashboard. Some users contributed localized version of this file and we put those files inside jide-properties.jar. If you want to support languages other than those we provided, just extract this properties file, translated to the language you want, add the correct postfix and then jar it back into jide-properties jar. You are welcome to send the translated properties file back to us if you want to share it.