1

Hello and welcome to this presentation of Building Diagram Displays for the Web using IBM ILOG JViews Diagrammer.

2

Prerequisites for this presentation are ... actually, there are many of them. "JViews Diagrammer Introduction;" "Building Diagram Displays Using the Diagrammer Tool Chain;" a suite of applications that comes with Diagrammer; "Building Diagram Displays for the Desktop" is also a prerequisite because we use the same components to produce our web views that we would use in a live desktop application.

And there is another, somewhat optional presentation. "JavaServer Faces Support in JViews" will help understand how to construct a project using JViews and JavaServer Faces.

The data files from those other presentations that are going to be used in this presentation are: nodes.jar which is a palette of symbols used to populate the diagram and the dashboard. AcmeNetwork.idpr is a Designer project which defines the graphics for the main view you see at the [upper-middle] of this slide. AcmeNetwork.xml is a data file with static data to populate the diagram. It is one of many ways to populate a diagram. And AcmeNetwork.css is a stylesheet taht tells the diagrammer component what to draw and how to draw the individual things in response to the data it finds in the data set.

And then lastly there is a dashboard definition in the lower left of this view called AcmeNetwork.idbd that's not specifically covered in any of these presentations, but it's covered quite well in the user manual -- that is, how to create a dashboard definition.

We'll reuse one class from our desktop application presentation. And that file is DashboardUpdater.java.

3

In this presentation we're going to show you a demonstration of the sample we want to create. And we'll show you a quick snapshot of the JSF application structure. If you find that too short, please refer back to the JSF presentation prerequisite mentioned in the previous slide.

And then we'll talk about two important tags for creating diagrams and dashboards in a JSF web page. First will be a simple example showing you sort of a bare-bones approach to doing this. Then we'll show a slightly more dynamic and elaborate example that uses a JSF backing bean or managed bean in other words.

Then we'll talk about the other tags used to populate the view, the toolbars, and enable user interactions.

So let's see a demonstration.

4

Here we have a web page showing the same diagram, that we had in the desktop application. But we have no java running on the client side of this connection. As you just saw when it flashed b, we have a Tomcat server running on this machine. But that server could be running anywhere.

This diagram has a lot of functionality. I can zoom in with the three zoom buttons. I can also drag-zoom with this rectangle tool. When I do so, we can see that the overview in the upper left works the same as it did in the desktop version. I can refocus the main view just by dragging the black rectangle.

When I am zoomed in I can also pan the main view just by dragging around. Actually by default it will restrict its pan to the visible area of the graph. But you can override that.

Here I can click a button to zoom to fit.

At the [lower-left] we have our dashboard with the same gauges that were in the dashboard in the desktop application. My main point here is that you can reuse the dashboard definition on the web much as you would use it on the desktop.

Here though, we are only driving the top gauge. The bottom gauge is showing data about the data that's driving the main view. However, there is no way to change that. So none of those values changes in this sample. But we wanted to emphasize the reuse of both the views and some of the back-end code for both examples.

5

So let's go back to the presentation and we'll see how this application was created.

This is a good snapshot of the files involved in creating a diagram application for the web. You'll have one or more web pages. We have two, diagramDisplayWeb.jsp and simple.jsp. We'll show you those later.

We have a data directory containing the data files, just described.

We also have an images directory. This contains images that are used to populate the toolbar buttons that we're going to discuss in this presentation.

And then, of course, the WEB-INF directory that is common to all Java server, or Java web tier, applications. It contains a web.xml. That file defines the web application. But this one also contains faces-config.xml which defines some connections that are there to enable JavaServer Faces to drive the view from the server side Java components.

Then we have a number of different libraries. This is a very standard set, in the middle here where it says JSF Support, that you'll need for all [JavaServer Faces] applications. The build files int he sample though will handle this for you. It will pull these files out of the JViews distribution to create your [war] file for you.

And there is also one very special file here in the WEB-INF/lib directory called nodes.jar. That's specific to our application. It contains the symbol palette. It is very critical that you make sure it's on the classpath at runtime for your application or you'll get errors indicating that it is unable to draw those symbols.

One tip I'd like to share: When you're working with the JViews samples for JSF, or for any of the web applications, it's useful to see the deployed structure of the application. The samples come in a directory structure that's organized differently than you'll actually see in the deployed version of the app. So what I like to do is go to the Tomcat server that's supplied with JViews. and just look through the webapps directory there to see how these deployed apps look at runtime.

You can also do something else. When you're building one of these samples in the JViews distribution for use on a web server that's not the one that comes with JViews. You'll want to use a different target for building them. There's a build.bat and build.sh file in those directories. Those batch files will build the version that's usable in the JViews distribution because that distribution links the Tomcat server to the JViews jar files that are needed. So you don't have to add those to your web app if you're deploying on the server that comes with JViews.

However, in a bare-bones Tomcat server, you'll need all of those jar files. For that purpose we'll use the build.complete target with ant. The ant script that comes with each of these samples.

6

Now let's look at some code. The simple example -- the simplest sort of diagram you can put on the screen looks like this. We have these declarations at the top that invoke the tag libraries that are provided by JViews [or] by JSF. So that we're able to write JSF tags in our web page and also reference JViews components with those tags.

The beginning here is just an HTML opening. And then we have <f:view. That's a JSF tag to begin a JSF view at this point. Then we have a JViews tag, jvdf:diagrammerView. This comes from the JViews Diagrammer library. We have a lot of different attributes we can play with, but the minimum set of these would specify some kind of data file -- in the case the idpr file which defines an entire view; and a width and height.

That populates the top half of the screen shot you see at the lower right of this slide. The bottom half is very similar. It would just load an idbd file instead of an idpr file.

7

Now for the full example. In the real world that's not enough for a really intricate application. So let's have something closer to a real-world example.

In the first place we do have the tag library declaration at the top. The rest of the page would fill in between these two tags. Then we have our jvdf:diagrammerView tag again. We give it an ID. This is

useful because we'll want to connect to it from other tags or from javascript - we might want to reference it. So that ID is critical.

The second line is the most interesting. We're creating a link between this tag and a backing bean on the server. The proper name for a backing bean is a "managed bean" - a "JavaServer Faces managed bean."

Somehow this web page has to know how to go find this Java bean. The Java bean name in terms of JSF is [sampleBean] and we're invoking something on that bean called "diagrammer." That means there is a java class somewhere with a method called "getDiagrammer." In the next chart I'll show you how the connection is made.

We also specify a width and height. We enable an interactor - I'll talk about those in a moment. Right here we're specifying the idpr file again. Even though we specify a diagrammer [reference] above we can still load it with the tag attribute in the web page.

We give a little GIF file for animated wait cursor. We also want to specify a message box ID so any output from this component can be displayed to the user as status -- it's like a status box.

And we set the background color to white.

8

So now -- coming down here - that sample bean name that was connected to the diagrammer instance for the tag we saw, is mentioned in the file called faces-config.xml.
There are a lot of things you can do with faces config, but the bare bones minimum to get a really dynamic web application going is to specify a managed bean. So we're doing that here. We give it a name -- anything we want -- but we have to use that name in our jsp pages.

When we mention that name, we're really talking about this class. Here is a whole package name and class, sampleBean. So that means that somewhere in our Java source code we have faces.dhtml.SampleBean.java.

In fact, in that class, because the JSP page [uses] sampleBean.diagrammer, we know there is a getDiagrammer method in the SampleBean.java. Here it is.

In this case, we're simply returning a diagrammer to which we've retained a reference -- because we're going to use that elsewhere.

We might also want to use this to create data on the server side. That's a more common scenario: you're loading data dynamically from a database or some kind of cross-process call .. who knows. You would want a reference to the diagrammer so you can get the data model from it as we've done here in this commented-out code near the bottom of the slide. Using that model, you can do whatever we would do with the model -- things like we did in the desktop sample in the previous presentation.

Let me back up here a moment - this bean is session scoped. So we're just retaining the diagrammer instance as a field in the class. So every instance of that class will be ... will contain one reference to a

diagrammer. There are other options here, but that's the most common scenario. BUt that's why the diagrammer is stored as a field in the SampleBean class.

9

The dashboard view is very similar to the diagram view. In fact it's nearly identical. It extends the diagramView classes in javascript. So it has all of the same attributes. It just takes a couple of different attributes. Dashboard diagram is what we're assigning to the dashboard element in our backing bean. So that will be very similar to what we just saw for the diagrammer.

And that's about it. Everything else is the same as for the diagramView .

10

Here is the getDashboard method. But we're doing something special here. Here we're using a reference to the diagrammer to create a DashboardUpdater. So if we wanted to create one of these just like we did in the desktop application, this is how we would do it. We can pass in .. we can use the diagrammer to get its data model and pass that into the DashboardUpdater.

I'm not going to go into great detail about the DashboardUpdater because that was covered in the desktop application. But this is ... rather simplistic but a *good* use though ... for retaining references to the diagrammer and dashboard on the server side.

11

Other things we'll want to add to the web application include toolbars like you see here. And each of those buttons on that toolbar is an imagebutton [provided by JViews]. You don't have to use the JViews classes for this, but they are convenient. And the samples that come with JViews use them. So we're discussing them here.

We first declare another tag library. It's JViews core ... we call it. "Jv" is the typical prefix we assign t it. It can be anything you want, of course. And this image button lets us set an image for different states of the button -- when it's rolled over, when it's selected, and the default -- is in this case, zoomrect.gif.

We can define the [button] action. In this case we're setting an interactor. I'll show you what an interactor looks like in just a moment. You can assume that this thing called, "zoom" is defined elsewhere in the page.

12

Pan and zoom and overview are provided in this other tag library, [correction: should be JViews Framework tag library].

Here's a tag that defines an overview with width and height. We have to pass in the diagrammer ID to this element so it knows what diagram it's trying to be an overview for.

And again we have this messages tag so we can output status information. And autoRefresh is true on this one.

13

Here are the interactors. We have a zoom and a pan interactor for this one. The other buttons on the toolbar were single event clicks. So when you clicked them, you were done with them. But the zoom and the pan interactors are click and drag operations so they require something more.

And this is the something more: it's a zoom interactor. We give it an ID. And then we use it elsewhere. e assign it using the buttons on the toolbar. Or as you saw with the diagram tag , we assigned it by default on the diagram view.

In summary, you'll follow these steps to build a JViews diagrammer web application. First use the symbol editor to define the symbols. Then use designer to assign those symbols to data objects that will be in the data model in memory at runtime. You can also use the dashboard editor if you want to define a dashboard view to add to your application.

Then you'll define the web pages. For that you'll use the JViews tag library for JSF. That's a separate javadoc that's accessible from the main JViews Diagrammer documentation. That's accessible as an index.html file in the root directory of the JViews Diagrammer portion of your installation.

Then you'll add a backing bean in Java to provide dynamic data and animation. Then you'll use JViews tags again to add user interactions to the web page.

If you want to extend this example with, for instance, popup menus, there's an example for that in jsf-diagrammer-gallery. That's a sample that comes with the JViews Diagrammer installation.

There are other samples, that one included, that show how to do any kind of custom action when the user clicks on the diagram itself -- and to know in the code for that custom action if any objects were under the click.

15

We wish you luck with your JViews Diagrammer project. Thank you for your time.